

Serverside programmering III

Dag 1 : Opret serverside web app.

Indhold:

1. Faglig mål	2
2. Øvelse (Case).....	3
Baggrund	3
Krav	4
Hvordan du bliver bedømmet	7

Faglige mål:

Dagens øvelse dækker følgende målpinde:

1. Eleven kan udvikle serverside webapplikationer, der kan levere HTML-kode til browseren, samt Web API eller webservices, som kan udveksle data med en client-application, f.eks. en browser eller en mobil App.
2. Eleven kan redegøre for forskellige arkitekturen for web Applikationer og web API (web Services), med fordele og ulemper.
3. **Eleven kan opbygge og konfigurere en web Application og web API (web service) vha. et framework.**
4. Eleven kan benytte validering af brugerinput i en web Applikation.
5. Eleven kan implementere passende ViewModels eller DTO klasser.
6. Eleven kan anvende Unit Test og mocking af objekter.
7. Eleven kan konfigurere routing i en applikation.
8. **Eleven kan udvide en applikation med en database, evt. med et ORM-framework.**
9. Eleven kan programmere services til brug for en applikation, f.eks. data- og logging-services. r
10. Eleven kan benytte en hensigtsmæssig strategi for Exception handling.
11. **Eleven kan implementere sikkerhed og brugeradministration i en web applikation.**
12. Eleven kan udrulle (deploy) en applikation, både On-Premises og Cloud baseret.
13. Eleven kan udføre Parallel Programmering.
14. Eleven kan redegøre for fordele/ulemper ved forskellige teknikker inden for Cryptography.
15. Eleven kan anvende Hashing, Symmetric og Asymmetric Encryption.

Øvelse:

Øvelsen er selvstændigt, hvor du via instruktionerne (under afsnit ”krav”) selv skal undersøg og afprøve dine løsninger. Brug gerne underviser som vejleder, også til afdækning af teori løbende efter jeres behov.

Baggrund:

Til authentication og authorization **bør** man anvende et **predefineret sikkerhedsframework**, der er knyttet til det valgte programmeringssprog/framework til styring af login, roller og brugerkontrol i en server-side webapplikation. Frameworket bør gemme brugeroplysninger sikkert i en **normaliseret database** og kunne udvides med funktioner som to-faktor-autentificering, kryptering og eksterne loginudbydere (fx Google, Facebook m.fl.).

Hvorfor er et predefineret sikkerhedsframework at foretrække?

- **Standardiseret sikkerhed og vedligeholdelse**
Brug af et velforprøvet framework sikrer, at applikationen følger **etablerede sikkerhedsstandarde** og **bedste praksis**, hvilket gør det lettere at vedligeholde og auditere, end hvis man opfandt sin egen løsning.
- **Beskytter mod kendte angreb**
Predefineret sikkerhedsframework er designet til at modstå **kendte angrebstyper** såsom SQL Injection, XSS, CSRF, brute-force og session hijacking – som ellers kræver betydelig ekspertise og ressourcer at imødegå manuelt.
- **Tillid og dokumentation**
Det er lettere at **opnå kundetillid**, når man baserer sin løsning på anerkendte sikkerhedsstandarde, frem for at skulle bevise, at ens egen hjemmelavede løsning er lige så sikker – hvilket ofte er urealistisk uden certificeringer og penetrationstests.
- **Automatiske sikkerhedsopdateringer**
Frameworks er ofte integreret i større økosystemer, som modtager **løbende sikkerhedsopdateringer** via pakkehåndtering eller operativsystemets opdateringssystem. Ved egenudviklede løsninger skal man selv identificere og reagere på nye trusler – en **ressourcetung og risikabel** opgave.

Selvom man anvender et sikkerhedsframework, er det vigtigt at være opmærksom på, at **det ikke fritager udvikleren for ansvar** for den samlede sikkerhed. Visse følsomme data – f.eks. **CPR-numre, helbredsoplysninger eller finansielle data** – kræver særskilt beskyttelse, som ikke nødvendigvis er dækket af frameworkets login standardfunktioner.

- Et eksempel er opbevaring af CPR-numre, hvor man bør anvende **kryptering** (ikke kun hashing) og sikre, at adgangen til sådanne oplysninger er **begrænset**. Her er det nødvendigt, at udvikleren har **forståelse for kryptografiske principper, databaseskyttelse og relevant lovgivning** (f.eks. GDPR).

Krav:

Undervisningen er rettet mod C# .NET-platformen, som benytter **.NET Core Identity** som sit **foruddefinerede sikkerhedsframework (Se dens features på side 8)**:

1. Opret en serverside webapplikation med følgende sikkerhedsfunktionaliteter:

- 1.1. **Registrering:** Webapplikationen **SKAL** indeholde en registreringsside, hvor nye brugere kan oprettes.

Krav:

- Der **SKAL** være en database til lagring af brugerdata.
 - Databasen **SKAL** være normaliseret.
 - Passwords **SKAL** gemmes krypteret.
- Der **SKAL** implementeres krav til stærke passwords:
 - Minimum 8 tegn
 - Mindst ét tal
 - Mindst ét stort bogstav
 - Mindst ét specialtegn
- Registreringssiden **SKAL** validere alle inputfelter, f.eks.:
 - Brugeren findes allerede
 - Mismatch mellem password og bekræft password
 - Ugyldigt e-mailformat
 - Password opfylder ikke kravene
- Webapplikationen **SKAL** anvende 2-faktor login som andet trin i autentifikationen, f.eks. med Google eller Microsoft Authenticator.
- Efter registrering **SKAL** applikationen generere en QR-kode, som brugeren kan scanne med sin Authenticator-app.
 - (Alternativer som e-mail eller SMS-baseret 2FA gennemgås senere.)

⚠ Hvis ovenstående ikke er opfyldt, betragtes det som en væsentlig mangel.

- 1.2. **Login:** Webapplikationen **SKAL** indeholde en login-side, hvor registrerede brugere kan logge ind.

Krav:

- Login-siden **SKAL** validere alle inputfelter, f.eks.:
 - Brugeren findes ikke
 - Forkert password
 - Ugyldigt e-mailformat
- Efter korrekt indtastning af brugernavn og password, **SKAL** brugeren gennemføre 2-faktor login via Authenticator (f.eks. Google eller Microsoft Authenticator).

⚠ Hvis ovenstående punkter ikke er opfyldt, betragtes det som en væsentlig mangel.

- 1.3. **Roller:** Webapplikationens bruger registerings side **SKAL** indeholde et felt til håndtering af brugerroller (**Authorization**).

Krav:

- Tilføj en brugerrolle med navnet "**Admin**" via en rolle-side du selv implementer.
- Opret en ny bruger og tildel denne **Admin**-rollen.

- Opdater webapplikationens **Home-side**, så teksten "**You are admin!**" vises, når en bruger med Admin-rollen logger ind.

⚠ Hvis ovenstående ikke er opfyldt, betragtes det som en væsentlig mangel.

2. Tilføj en separat database: **TodoDb**.

Du skal oprette en separat database kaldet TodoDb til din webapplikation. Denne database skal indeholde følgende to tabeller: Cpr og Todolist.

Grund til, at du skal implementere en separate database og ikke anvende det eksisterende Identity database er: best practis konvention hvor du **IKKE MÅ** ændre eller udvide Identity database med kolonner eller tabeller, der ikke er 100% relateret til Identity-funktionalitet. TodoDb anvendes nemlig til data, der **IKKE** er relateret til brugerregistrering og autentifikation.

⚠ Hvis dette krav ikke overholdes, betragtes det som en væsentlig mangel.

TIPS: Her er et design eksempel på hvordan din TodoDb kan se ud:

Table	Column	Type	Allow Nulls
Cpr	Id	int	✓
	UserNr	nvarchar(200)	✓
Todolist	Id	int	✓
	UserId	int	✓
Todolist	Item	nvarchar(500)	✓

3. Opret Frontend-integration med TodoDb.

Din webapplikation skal have en **brugergrænseflade (frontend)**, som integrerer med den oprettede **TodoDb**-database.

Krav: Visninger (**Views**) skal fremstå **pæne og ordentligt strukturerede** – layout, overskueligt (må godt være minimalist, men pæn).

3.1 Adgangskontrol og navigation:

- Hvis en bruger ikke er logget ind, skal webapplikationen automatisk omdirigere til login-siden. Dette gælder uanset hvilken side man forsøger at tilgå – selv ved direkte URL-adgang i browseren.

⚠ Hvis dette krav ikke er opfyldt, betragtes det som en væsentlig mangel.

3.2 Kan man ikke login fordi man ikke er registeret, skal man kunne vælges at register:

To-do list app, login:

Email

Password

Log in

Remember me

3.3 Når der skal registeres ny bruger, skal der kunne angives en rolle, hvor rollen **SKAL** vælges med en "dropdown" element. Dette **SKAL** implementeres i din Identity register side:

To-do list app, register

Email
no2@tec.dk

Password

Confirm Password

Add Role
Select a role

Admin

Register

To-do list app, register

Email
no2@tec.dk

Password

Confirm Password

Add Role
Select a role

Admin

3.4 Opret en **cprNr** side. Efter login, **skal** applikation **ALTID** auto navigeres til denne cpr.nr. side, hvor brugeren **SKAL** angive sit Cpr.Nr. Man skal kunne se brugerens roller vist samman med bruger navn:

About no2@tec.dk Logout

Validate CRP.NR.

User: no2@tec.dk

Cpr-nr:

Submit

Denne bruger har ingen rolle.

About no2@tec.dk (Admin) Logout

Validate CRP.NR.

User: no2@tec.dk

Cpr-nr:

Submit

Denne bruger har ingen rolle. Denne bruger har "Admin" rolle.

Derefter implementer en side som når applikation køre, kan oprette roller i Identity databasen (Dette bør alleredet være implementeret fra øvelse 1).

3.5 Opret en **TodoList** side. Ved første gang log ind, **skal** cpr.nr. siden gemme det indtastet cpr.nr i databasen og applikation auto navigeres til TodoList siden. Har man haft sin cpr.nr. gemt i databasen, **skal** det gemt cpr.nr. hentes frem og **valideres** med det cpr.nr. brugeren angiver på CprNr siden og auto navigeres til TodoList siden hvis validering er bestået:

Hvis cpr.nr. er forkert, skal to-do item side ikke vises, men i Stedet en fejl besked:

To do list

Add to-do item:

Submit

My to do items

test 1

test 2

Validate CRP. NR.

User: no@tec.dk

Cpr-nr:

Submit

Cpr-nr er forkert!

TodoList side som behandler to-do items, hvor hver opret items **gemmes** i **TodoListtabellen** i **TodoDB** databasen under login brugeren.

Hvordan du bliver bedømmet

- Du skal kunne køre/fremvis din applikation fejlfrit og med alle funktionaliteter fra foroven beskrevet punkt 1 til 3 samt deres underpunkter (brug punkterne 1 til 3 som tjeklist).
- Du skal kunne identificerer hvor i din implementation der krypteret er lageret. Samt hvilket teknik indenfor kryptografi er anvendt (hashing eller kryptering).

.NET Core Identity features

Feature	.NET Core Identity
User Management (CRUD, roles)	Identity
Authentication	Identity + Middleware
Authorization (Roles/Claims)	Identity + Policies/Roles
Password Hashing	Built-in (Identity uses PBKDF2 by default)
Two-Factor Authentication (2FA)	Identity 2FA + QR Code support
Login with External Providers	Identity + Google / Facebook / Microsoft / Twitter / OAuth / OpenID Connect
Session or Token-Based Authentication	Identity Cookie / JWT
Input Validation	Identity validation + Data Annotations
Account Lockout	Built-in (after configurable failed login attempts)
Password Policies	Configurable (min length, required characters, lockout thresholds, etc.)
Email Confirmation	Supported via Identity + Token Providers (email confirmation links)
Phone Number Confirmation	Supported via Identity + Token Providers (SMS or external SMS gateway integration)
Password Reset	Identity supports token-based password reset
Security Stamp Validation	Built-in (invalidates cookies when critical changes are made)
Claims-based Identity	Supported (store and evaluate claims for fine-grained authorization)
Role-based Identity	Supported (users can belong to roles, used in policies/authorization)
Custom User Properties	Extend IdentityUser class to include additional fields
Dependency Injection Integration	Fully integrated with ASP.NET Core DI container
Entity Framework Core Integration	Default with EF Core (backed by SQL Server, SQLite, PostgreSQL, etc.)
Pluggable Stores	Swap EF store with custom stores (e.g., MongoDB, Cosmos DB, etc.)
GDPR Support	Identity supports data protection APIs for compliance (user data download/delete)
Logging / Diagnostics	Integrated with ASP.NET Core logging + events