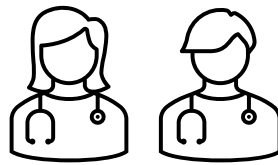


11/30/2022

Symptomizer

Version 2



ITPE3200 Web Application
GROUP 8

Denne prosjektet er laget ved Groub 8 studenter:

Student Navn	Student Nummer
Aleksander Korkh	s354398
Aya Abdelhady	s354367
Jessica Chackyan	s351930

NB!:

Angular prosjekt er laged ved Asp.net core 3.1 og

Node version 16.10.0

CONTENTS

<i>prosjekt valg</i>	3
<i>Introduction</i>	3
<i>hvordan bruke prosjektet</i>	4
<i>Symptomizer Version 2</i>	4
<i>Klient side: Client app</i>	5
<i>Server side: Models</i>	10
<i>Server side: Controllers</i>	10
<i>Server side: dal</i>	12
<i>Tester</i>	14
<i>Refrences</i>	19

PROSJEKT VALG

Symptom til diagnose kalkulator.

En løsning hvor man kan velge forhånds definerte symptomer og løsningen skal fortelle deg hva du har (dvs. din diagnose).

Funksjonalitet:

- CRUD Funksjonalitet
- Server/Client- inputvalidering.
- Innloggingsmekanisme for brukere, med sikkerhets mekanismer som hashing og salting.
- Enhetstester på server.
- Feilhåndtering og Logging.
- Single Page Application (SPA) via enten react eller angular.

INTRODUCTION

Denne applikasjonen representerer en fungerende prototype av prinsippene som kan brukes til å utvikle en ekte nettassistent som kan bidra til å stille korrekte diagnoser basert på pasient symptomer. I vår søknad vil vi bare beholde 3 sykdommer (diagnoser) og et sett med forhånds definerte symptomer. I et stort prosjekt kan det lages en enorm database med alle kjente symptomer og sykdommer. I henhold til oppdragskrav ble applikasjonen opprettet ved hjelp av ASP.NET Core 3.1, Web App ved bruk av, C#, Angular og Typescript.

Prosjektet er forlengte av den forrige Versjon som heter Symptomizer. Det gir mulighet til pasient å logge in på en nettside etterpå når alt er suksess, mottar det pasientnavn (fornavn og etternavn) og representere en liste i form av sjekkboks hvor pasient kan velge hva slags symptoms han/hun har, og den skal kalkulere hva slags sykdom den personen har ved å bruke forhåndsdefinerte data. Prosjektet støtter CRUD funksjon hvor pasient kan endre eller slette sitt resultat fra database.

Prosjektet har det samme back-end som det hadde fra forrige versjon med noen modifikasjon som skal bli representert senere i denne dokumentasjon.

I tillegg, prosjektet inkludere tester som heter PatientTester som er bygd ved bruk av xUnit templates.

HVORDAN BRUKE PROSJEKTET

For å kjøre prosjektet vårt krever det Visual Studio (Microsoft) eller Rider (JetBrains) å være installert.

Prosjektet er lagt opp som Zip fil. Du trenger å trekke ut fillen og etterpå du kan kjøre Appen ved å klikke på Symptomizer_v2.sln. Du må bruke Visual Studio for å kjøre Koden.

Om solusjon åpner ikke, prøv å gå inn filen
Symptomizer_v2→Symptomizer_v2.csproj.

Du kan kjøre koden ved å trykke på den Grønne knappen i Visual Studio i navigasjon bar eller gå inn Debug→Start without Debugging.

SYMPTOMIZER VERSION 2

Prosjektet består av to forskjellige deler:

klient side som kan finnes i Client App → src → app og der vi har lagde fem forskjellige mapper [add, edit, home, login, menu] som organiserer front end visualisering.

Klient side har også Routing Component for å flytte mellom de forskjellige sider.

Server side består av tre mapper [Controllers, Models, DAL]

KLIENT SIDE: CLIENT APP

På klient Side vi har lagde 5 separert mapper: [add, edit, home, login, menu].

hver mappe bestå av to filer html og typescript, bortsatt fra home mappe hvor det ligger 4 filer, de to ekstra filer er for delet modal som ble opprettet fra forelesning notater

Menu: inneholder programmet navn : Symptomizer med en navigasjon bar hvor den har ruter til Home side, Add side eller log ut. Etter du logger ut, får du ikke tilgang til hjemmetsidet heller ikke til lagre en ny pasient side.

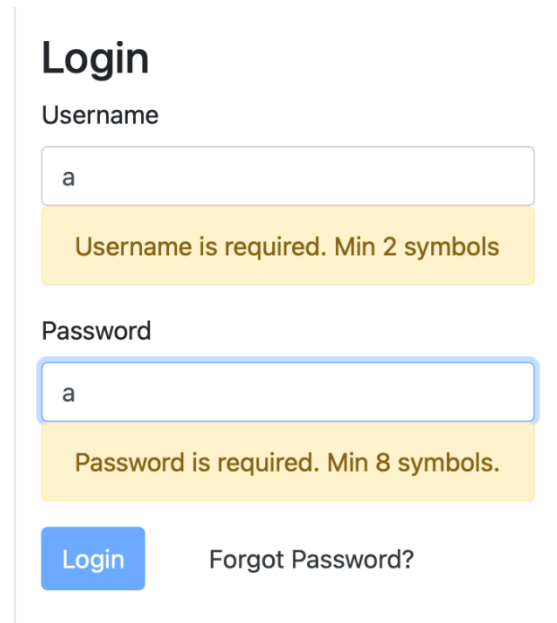


I tillegg til foter som inneholder vårt gruppa tilte.



Login: det viser enkelt logge inn side med brukernavn og passord feltet hvor brukeren kan logge på med forhåndsdefinert data. Det ble lagt til validering av input slik at feltet kan ikke være tomt og at passord kan ikke være mindre enn 8 tegn og brukernavn kan ikke være mer enn 30 tegn. Selvfølgelig de spesiell norsk tegn ble lagt til om person har det i navnet sitt.

Når logge på lykkes, brukerne ble overført videre til hjemmet side.



Login

Username

Password

[Login](#) [Forgot Password?](#)

Username: Admin

Password: Admin123

Username : Admin

Passord: Admin123

Pass på at begge brukernavn og passord skrives med store bokstaver i begynnelsen.

Programmet gir deg mulighet til å sjekke hva er passord og brukernavn om du har glemt det.

Home: Etter at logge på ble lykket, får du tilgang til hjemmet side hvor det representere en SPA (Single-Page-Application) via Angular.

Liste viser data til to forhån definerte pasienter med sine symptomer og symptoms diagnose de har fått fra database.

List of patients:

First name	Last name	Symptoms	Disease		
Ole	Hansen	Fever or chills,Cough,Sore throat,High temperature,Muscle or body aches	Flu	Delete	Edit
Per	Jensen	Fever or chills,Cough,Sore throat,High temperature,Shortness of breath or difficulty breathing,Muscle or body aches	COVID-19	Delete	Edit

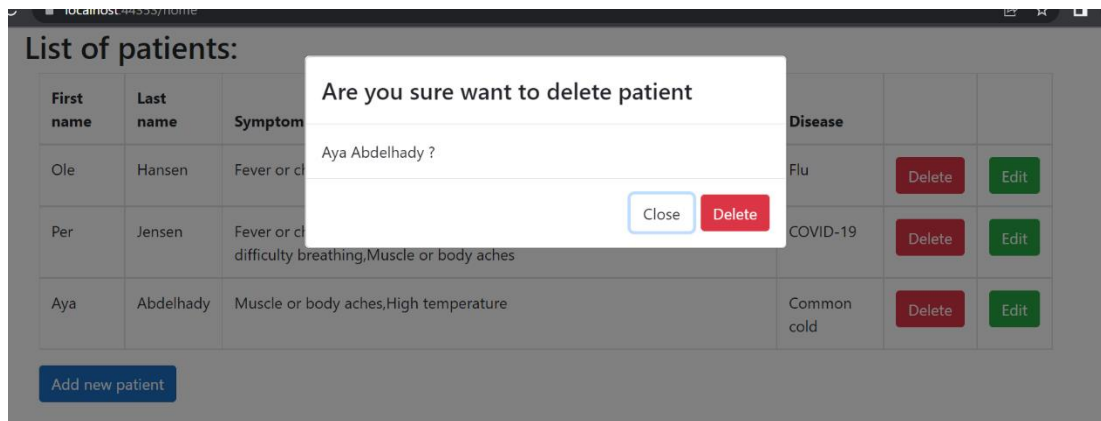
[Add new patient](#)

Som du ser i bildet oppe programmet støtter CRUD funksjon hvor hver pasient har mulighet til å slette eller justere de symptomer han har valgt.

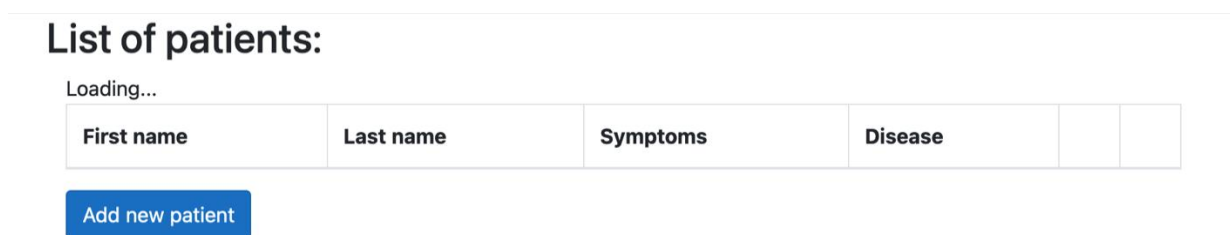
Når du klikk på (Add new patient) du ble omredigert til et annet nettside hvor du kan lage en ny pasient, mer om dette kommer i Add del.

For å gjøre knappen Delete aktiv, vi har lagde delete-modal i home mappe og har brukt det samme kode fra forelesning.

Når du trykker på knappen, du skal få en melding tilsendt fra console med to valg enten godkjenne fjerning, eller unngå handlingen.



NB! du kan åpne hjemmeside ved å justere lenken i søk feltet slik <https://localhost:????/home> da du hoppe over å logge på og gå rett til hjemmeside, men du får ikke lov til å se data (pasient liste vi har allerede lagde) heller ikke lage en ny pasient om du ikke har loget på.



Add: brukeren skal se et skjema hvor han/hun må skrive sitt navn og velge sine mest likne symptomer fra en liste i form av sjekkboks.

For å lage sjekkboks i angular, har vi brukt for løkke i html side som går gjennom et array av symptoms definert i typescript side.

Patient's Credentials

First name

Per

Last name

Jensen

What symptoms do you have?

☐ Fever or chills

☐ Cough

☐ Sore throat

☐ High temperature

☐ Shortness of breath or difficulty breathing

☐ Muscle or body aches

Minimum 1 checkbox value is required.

List of symptoms before correction:

• Fever or chills

• Cough

• Sore throat

• High temperature

• Shortness of breath or difficulty breathing

• Muscle or body aches

Apply changes

Cancel

Input feltet har validering funksjonalitet som er det samme som vi hadde i brukernavn i å logge inn side.

Når alt er klart, pasienten kan trykker på Register knapp hvor han/hun blir omredigert videre til hjemmet side til å se resultater av sine valgte symptomer.

List of patients:

First name	Last name	Symptoms	Disease		
Ole	Hansen	Fever or chills,Cough,Sore throat,High temperature,Muscle or body aches	Flu	Delete	Edit
Per	Jensen	Fever or chills,Cough,Sore throat,High temperature,Shortness of breath or difficulty breathing,Muscle or body aches	COVID-19	Delete	Edit
Aya	Abdelhady	Fever or chills,Cough,Sore throat	Common cold	Delete	Edit

Add new patient

Edit: om pasient har valgt noen symptomer ved feil, programmet gir bruker mulighet til å justere dette ved å klikke på den grønne knappen Edit.

Dette skal omredigere bruker til et nytt nettside hvor han/hun skal ha det same (Add new patient) skjema med liste som viser hvilket symptom de har valgt og gi dem mulighet til å justere info slik som det er beskrevet i bildet nedover.

Aya

Last name

Abdelhady

What symptoms do you have?

☐ Fever or chills

☐ Cough

☐ Sore throat

☒ High temperature

☐ Shortness of breath or difficulty breathing

☒ Muscle or body aches

List of symptoms before correction:

Fever or chills

Cough

Sore throat

Apply changes

Cancel

Når pasient er fornøyd med endring, han/hun gjorde, han kan trykke på Apply changes for å se det nytt resultatet som programmet skal kalkulere.

SERVER SIDE: MODELS

I tillegg til patient klass vi har lagd en bruker klass som inneholder string of brukernavn og passord med sitt get og sett metoder.

```
2 using System.ComponentModel.DataAnnotations;
3
4 namespace Symptomizer_v2.Models
5 {
6     public class User
7     {
8
9         [RegularExpression(@"^[a-zA-ZæøåÆØÅ. \-]{2,20}$")]
10        public String Username { get; set; }
11        [RegularExpression(@"^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$")]
12        public String Password { get; set; }
13    }
14 }
```

SERVER SIDE: CONTROLLERS

Controllers har i utgangspunktet de samme funksjoner fra forrige versjon med små endringer som er lagt til som logg og sesjon, i tillegg til logg inn metode. Siden er prosjektet bygget i Angular, så trenger Controllers Rest Statetransfer til å kommunisere med API på server. for å bruke disse kallene så må vi dekorere metoder med sitt kall.

LoggIn metode: metoden skal bruke loggIn funksjon som ble definert i PatientRepository. Først det skal sjekke om brukernavn og passord input er valid, hvis det er så, returnere true status og gi brukeren tilgang til å bruke programmet videre.

```

[HttpPost("login")]
public async Task<ActionResult> LoggIn(User user)
{
    if (ModelState.IsValid)
    {
        bool returnOk = await _db.LoggIn(user);
        if (!returnOk)
        {
            _log.LogInformation("Failed to logg in with user");
            HttpContext.Session.SetString(_loggedIn, _NotLoggedIn);
            return Ok(false);
        }
        HttpContext.Session.SetString(_loggedIn, _loggedIn);
        return Ok(true);
    }
    _log.LogInformation("Fail in input validation");
    return BadRequest("Fail in input validation");
}

```

Log vil gi oss muligheten til å vite alt som skjer i bakenden, [når hver eneste metode fra kontrollen ble kalt] siden programmet begynner å kjøre og kommunisere med serveren ved å holde styr på all tilkoblingsutveksling i form av tekstfil som vil bli opprettet når tilkoblingen er etablert og programmet starter løping.

Filen lages i startup ved hjelp av loggerFactory-syntaks og lagres i Logs/Patientslog.txt.

Session: er for å sjekke om pasienten har logget på og i så fall kan programmet starte med å lede brukeren til hjemmesiden og gjøre de andre metoder gyldige ellers vil ikke brukeren ha tilgang til programmet. Sesjon vil også logge ut pasienten fra nettsiden hvis ingen handling er registrert innen en maksimal periode på 30 minutter.

SERVER SIDE: DAL

DB_Init.cs: vi har opprettet en forhåndsdefinert bruker, deretter brukte vi funksjonene fra pasientlageret til å hash og salt passordet.

```
// Creating predefined user
var user = new Users
{
    Id = 1,
    Username = "Admin"
};
var password = "Admin123";
byte[] salt = PatientRepository.CreateSalt(); //can be used var instead
byte[] hash = PatientRepository.CreateHash(password, salt); //can be used var instead
user.Password = hash;
user.Salt = salt;
context.Users.Add(user);
```

I **IPatientRepository**-grensesnittet har vi deklartert **LoggInn** funksjonen og i **PatientRepository**-klassen har vi definert funksjonen med ønsket resultat, hvor funksjonen vil gå gjennom databasen for å sjekke om brukernavnet og passordet er kjent for systemet ellers vil brukeren vil ikke være i stand til å logge på, og selvfølgelig registreres alle disse trinnene og sendes til loggfilen **PatientLogs.txt**.

```
public async Task<bool> LoggIn(User user)
{
    try
    {
        Users foundUser = await _db.Users.FirstOrDefaultAsync(u => u.Username == user.Username);
        byte[] hash = CreateHash(user.Password, foundUser.Salt);
        bool ok = hash.SequenceEqual(foundUser.Password);
        if (ok)
        {
            return true;
        }
        return false;
    }
    catch (Exception e)
    {
        _log.LogInformation(e.Message);
        return false;
    }
}
```

I tillegg har vi definert log ut metoden som skal slutte sesjon og la brukeren gå ut av programmet.

```
    }

    [HttpPost("logout")]
    public void LoggOut()
    {
        HttpContext.Session.SetString(_loggedIn, _NotLoggedIn);
    }
}
```

Vi har også definert funksjonene CreateHash og CreateSalt ved å bruke samme kode fra forelesningene.

```
2 references
public static byte[] CreateHash(string password, byte[] salt)
{
    return KeyDerivation.Pbkdf2(
        password: password,
        salt: salt,
        prf: KeyDerivationPrf.HMACSHA512,
        iterationCount: 1000,
        numBytesRequested: 32);
}

1 reference
public static byte[] CreateSalt()
{
    var csp = new RNGCryptoServiceProvider();
    var salt = new byte[24];
    csp.GetBytes(salt);
    return salt;
}
```

PatientContext-klassen har samme data fra tidligere versjon med tillegg av Users-tabell med sin egen variabel for get og set-metoder.

```
public class Users
{
    public int Id { get; set; }
    public string Username { get; set; }
    public byte[] Password { get; set; }
    public byte[] Salt { get; set; }
}

public DbSet<Patients> Patients { get; set; }
public DbSet<Diseases> Diseases { get; set; }
public DbSet<Users> Users { get; set; }
```

TESTER

Tester er et separert prosjekt som er bygd ved bruk av xUnitTester, for å teste de alle mulige resultater som kommer fra de metodene som vi har definert i Controller klasse.

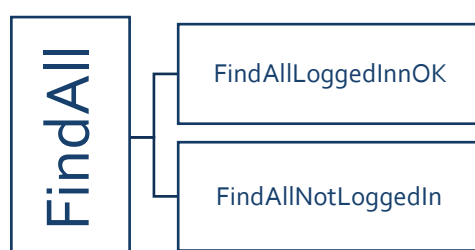
Siden prosjektet har sesjon, trenger dette flere Mock's, så lager vi en ekstra klass som heter MockHttpSession og definere det flere Mock'ene som trengs isteden å definere dem i den enkelte test.

vi har hentet MockHttpSession klass kode fra forelesningen.

Vi har behandlet de alle forskjellige mulige resultater senario som kan kommer for hver eneste metode, de kan spesifiser slik.

FindAll: tester to mulige utganger.

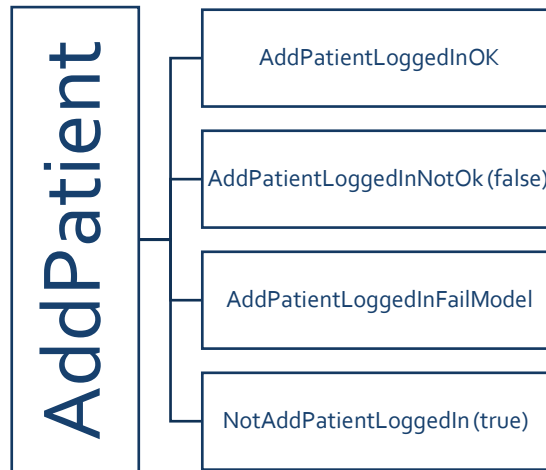
- Brukeren klarte å logge på så programmet skal liste all pasient fra database (FindAllLoggedInOK)
- Var noe feil med å logg på, og brukeren fått ikke tilgang til programmet. (FindAllNotLoggedIn)



AddPatient: behandle fire mulige utganger.

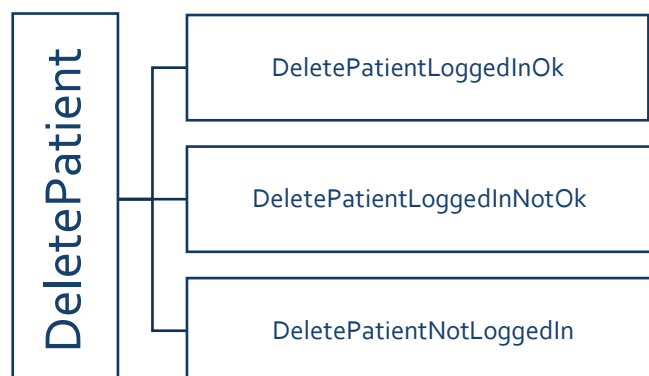
- Brukeren klarte å logge på derfor, fikk han/hun lov til å bruke programmet (AddPatientLoggedInOk)
- Brukeren klarte ikke å logge på derfor, fått ikke tilgang til programmet, kan ikke gå videre (AddPatientLoggedInNotOk)

- Brukeren klarte å logge på, og har skrevet feil i add skjema (AddPatientLoggedInFailModel)
- Brukeren klarte å logge på, men klarte ikke å lagge en ny pasient (NotAddPatientLoggedIn)



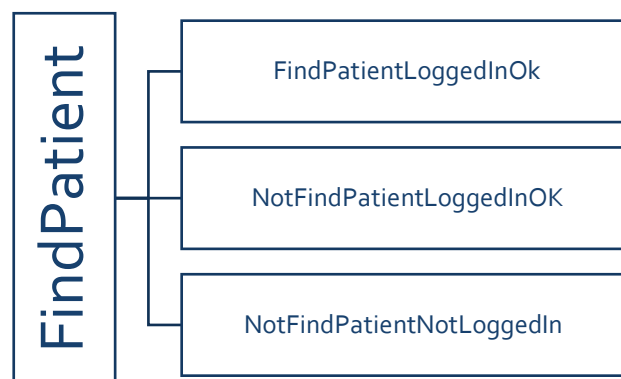
DeletePatient: behandle 3 mulig utganger:

- Brukeren klarte å logge på og slette info om pasient (DeletePatientLoggedInOK)
- Brukeren klarte å logge på, trykket på delete men unngå dette med å lukke fane isteden å konfirmer. (DeletePatientLoggedInNotOk)
- Brukeren kunne ikke logge på derfor, fått ikke lov til å slette. (DeletePatientNotLoggedIn)



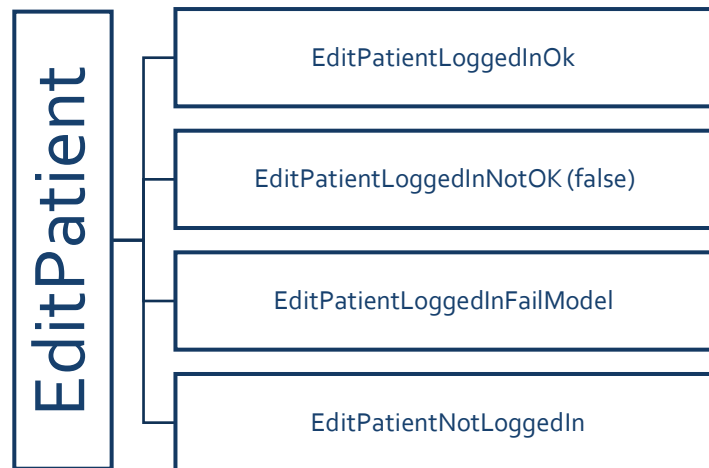
FindPatient : den metoden skal kunne hente en pasient etter sin id. De behandler tre mulige utganger:

- bruker har lyktes med å logge på og klarte å finne ønsket pasient (FindPatientLoggedInOk)
- Brukeren klarte å logge på, men fant ikke pasient. (NotFindPatientLoggedInOk)
- Brukeren klarte ikke å logge på. (NotFindPatientNotLoggedIn)



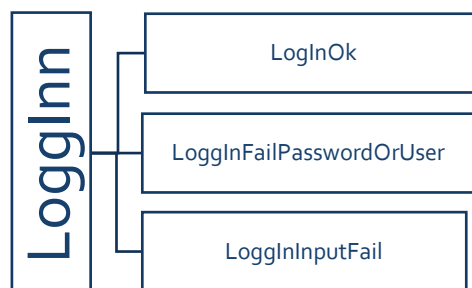
EditPatient: behandle fire mulige utganger:

- Brukeren klarte å logge på, og skrevet riktig i endring skjema uten noe feil i input [godkjent validering]. (EditPatientLoggedInOk)
- Brukeren klarte å logge på, men endret ikke noe i endring skjema (EditPatientLoggedInNotOK)
- Brukeren klarte å logge på, men skrevet feil i input feltet i endring skjema. (EditPatientLoggedInFailModel)
- Brukeren klarte ikke å logge på derfor kunne ikke å gå videre med programmet. (EditPatientNotLoggedIn)



LoggInn: behandler tre mulige utganger:

- Brukeren klarte å logge på (LogInOk)
- Brukeren finnes ikke, feil brukernavn eller passord (LoggInFailPasswordOrUser)
- Brukeren har gitt ikke gyldig tegn i logg inn felter (LoggInInputFail).



Og har vi ikke glemt å teste log ut metode med en enkelt test.

```
[Fact]
public void LoggOut()
{
    var patientController = new PatientController(_mockRep.Object, _mockLog.Object);

    _mockHttpContext.Setup(s => s.Session).Returns(_mockSession);
    _mockSession[LoggedIn] = LoggedIn;
    patientController.ControllerContext.HttpContext = _mockHttpContext.Object;

    // Act
    patientController.LoggOut();

    // Assert
    Assert.Equal(NotLoggedIn, _mockSession[LoggedIn]);
}
```

Til slutt, de alle tester ble kjørt og suksess.

Test	Duration	Traits
▲ SymptomizerTestProject (20)	219 ms	
▲ SymptomizerTestProject (20)	219 ms	
▲ UnitTest1 (20)	219 ms	
✓ NotFindPatientNotLoggedIn	2 ms	
✓ NotFindPatientLoggedInOk	3 ms	
✓ NotAddPatientLoggedIn	1 ms	
✓ LoginOk	1 ms	
✓ LoggOut	5 ms	
✓ LoggInInputFail	1 ms	
✓ LoggInFailPasswordOrUser	4 ms	
✓ FindPatientLoggedInOk	2 ms	
✓ FindAllNotLoggedIn	2 ms	
✓ FindAllLoggedInOK	2 ms	
✓ EditPatientNotLoggedIn	3 ms	
✓ EditPatientLoggedInOk	2 ms	
✓ EditPatientLoggedInNotOK	1 ms	
✓ EditPatientLoggedInFailModel	1 ms	
✓ DeletePatientNotLoggedIn	1 ms	
✓ DeletePatientLoggedInOk	175 ms	
✓ DeletePatientLoggedInNotOk	1 ms	
✓ AddPatientLoggedInOk	1 ms	
✓ AddPatientLoggedInNotOk	1 ms	
✓ AddPatientLoggedInFailModel	10 ms	

REFERENCES

<https://www.cdc.gov/coronavirus/2019-ncov/symptoms-testing/symptoms.html>

https://oslomet.instructure.com/courses/24253/pages/sessions?module_item_id=452360

https://oslomet.instructure.com/courses/24253/pages/enhentstest-av-kundeapp-logginn-losning?module_item_id=452362

https://oslomet.instructure.com/courses/24253/pages/kunde-spa-med-routing-og-modal-for-sletting?module_item_id=452375

https://oslomet.instructure.com/courses/24253/pages/kunde-spa-med-routing?module_item_id=452374

<https://angular.io/guide/http#requesting-non-json-data>