SWFED Mundtlig Eksamen

Forberedelse: Gennemgang af læringsmål

Indhold

MAUI

React

1 Redegøre for principperne i .Net frameworket og dets overordnede arkitektur samt beskrive og anvende programmeringssproget C#. (MAUI)

.NET arkitektur (Lag)

- Kode til app:
  C# creating models, view models, and services
- .NET MAUI API:
  (UI-elementer som controls og layout panels)
- Platform specifikke API'er

  (Oversætter .NET API til de enkelte platforme, fx android, ios, windows and macOS)

- .NET BCL (Bass class library):

  ObservableColelction, asynchronous programming.

- Mono Runtime og WinRT
  (eksekverer .net koden til det target platform)

1.1.1 Hvordan .NET applikationer kører og kompileres

- **Kompileringsproces**:
  - **C# kode**: Du skriver din applikationskode i C#.
  - **C# compiler**: Koden kompileres af C#-kompilatoren (csc.exe) til Common Intermediate Language (CIL), som er en lav-niveau, platform-uafhængig instruktionssæt.
  - **Assemblies**: CIL-koden lagres i assemblies (.dll eller .exe filer).
- **Kørselsproces**:
  - **Just-In-Time (JIT) kompilation**: Når en .NET-applikation køres, oversættes CIL-koden til native maskinkode af JIT-kompilatoren. Denne proces sker ved runtime, hvilket gør det muligt for applikationen at køre på den specifikke hardware og operativsystem.
  - **Common Language Runtime (CLR)**: CLR er .NET's runtime-miljø, som håndterer udførelse af kode, garbage collection, sikkerhed, undtagelseshåndtering, og andre systemtjenester.
- **Hukommelsesstyring**:
  - **Garbage Collection**: CLR inkluderer en garbage collector, som automatisk frigør hukommelse, der ikke længere er i brug. Dette hjælper med at forhindre hukommelseslækager og optimere hukommelsesforbruget.

Hvad skiller C# sig fra andre sprog? (GC)

- Marker
  Laver en liste af de forskellige referencer og følger rutes og tjekker på hvis et objekt peger på noget. Compact fase, hvis der er noget som ikke peger på noget, så kan det slettes
- De forskellige hukommelse i en computer
  (Program, CPU, static, stack og heap)
- Garbage collection (se slides fra lektion 5)
  Arbejder med meory i heap

1.1.2 Fokus på Garbage Collection i C#

Garbage collection (GC) er en automatiseret hukommelsesstyringsfunktion i .NET. Den administrerer tildelingen og frigivelsen af hukommelse for dine applikationer. GC opererer ved at:

- **Automatisk spore objekter**: GC holder styr på hvilke objekter i heap-hukommelsen der stadig er i brug og hvilke der ikke er.
- **Fjernelse af ubrugte objekter**: Når GC opdager objekter som ikke længere er tilgængelige (dvs. ingen referencer peger på dem), frigør den hukommelsen allokeret til disse objekter.
- **Genanvendelse af hukommelse**: Hukommelsen som er frigivet af GC kan genbruges til nye objekter.
- **Minimering af hukommelseslækager**: GC hjælper med at forhindre hukommelseslækager ved at sikre at alle ubrugte objekter fjernes og deres hukommelse frigøres.
- **Generational Collection**: GC bruger en generationel tilgang, hvor objekter der har overlevet flere GC-cyklusser flyttes til en højere generation, hvilket optimerer performance.

2 Anvende kontroller og layout panels til opbygning af grafiske brugergrænseflader. (MAUI)

For at opbygge en grafisk brugergrænseflade i MAUI, anvender man forskellige kontroller og layout panels. Nogle af de mest anvendte elementer inkluderer:

- **Controls**:
  - **Button**: En trykknap som brugeren kan interagere med.
  - **Label**: Viser tekst.
  - **Entry**: En tekstboks til brugerinput.
  - **DatePicker**: En kontrol til at vælge datoer.
  - **TimePicker**: En kontrol til at vælge tidspunkter.
  - **Editor**: En multi-line tekstboks til brugerinput.
  - **CollectionView**: En kontrol til visning af samlinger af data.
- **Layout Panels**:
  - **StackLayout**: Arrangerer børneelementer i en enkelt række eller kolonne.
  - **Grid**: Arrangerer børneelementer i et fleksibelt gitter.
  - **AbsoluteLayout**: Arrangerer elementer ved hjælp af absolut positionering.
  - **FlexLayout**: Arrangerer elementer i en fleksibel container med mulighed for at tilpasse layoutet.

Ved at kombinere disse kontroller og layout panels kan man skabe komplekse og interaktive brugergrænseflader i MAUI-applikationer.

3 Anvende navigering mellem pages i en applikation. (MAUI)

3.1 Teori

3.1.1 Shell and Route-Based Navigation in MAUI

Shell in .NET MAUI provides a simplified way to describe the navigation structure of an application using XAML. It supports a URI-based navigation scheme, allowing navigation to any page using routes.

Shell Overview:

- Shell enables defining the overall structure of the app's navigation.
- Routes are registered within `AppShell.xaml` and `AppShell.xaml.cs`.
- Navigation is performed using these routes rather than direct references to page instances.

3.2 Implementering

3.2.1 Registrering af routes I AppShell

```
<ShellContent
    Title="Home"
    ContentTemplate="{DataTemplate views:MainPage}"
    Route="MainPage" />
```

```
public partial class AppShell : Shell
{
    public AppShell()
    {
        InitializeComponent();

        // Register routes
        Routing.RegisterRoute(nameof(CalendarPage), typeof(CalendarPage));
        Routing.RegisterRoute(nameof(InvoicePage), typeof(InvoicePage));
    }
}
```

3.2.2 NavigationService

Implementerer funktioner, som kalder navigation linjer

- ToPage with parameter: `await Shell.Current.GoToAsync($"{page}?{query}";);`
- ToPage: `await Shell.Current.GoToAsync(page);`
- BackToPage: `await Shell.Current.GoToAsync("..");`

3.2.3 Kalder på funktioner I navigationService I relaycommands

```
// Navigate to CalendarPage
[RelayCommand]
public async Task GoToCalendar()
{
    await _navigationService.NavigateToPage(nameof(CalendarPage));
}

// Navigate to InvoicePage
[RelayCommand]
public async Task GoToInvoices()
{
    await _navigationService.NavigateToPage(nameof(InvoicePage));
}
```

3.2.4 Passering af parameter og modtagelse

Passing from page:

```
1 reference
private async Task NavigateToInvoicePageAsync(Work selectedTask)
{
    if (selectedTask != null)
    {
        await _navigationService.NavigateToPage(nameof(InvoicePage), parameters: new Dictionary<string, object>
        {
            { "TaskId", selectedTask.Id }
        }); // Task
    }
}
```

Route Parameters

```
private async void Button_Clicked(object sender, EventArgs e)
{
    var product = new Product
    {
        Name = "RouteDemo",
        ManufacturerName = "Front-end(FED)"
    };

    var pageParams = new Dictionary<string, object>
    {
        {  "Product", product }
    };

    await Shell.Current.GoToAsync("productdetails", pageParams);
}
```

pass values

The page which registered route

Receiving at page:

Use **QueryPropertyAttribute** on the page that we want to receive parameters

```
[QueryProperty(nameof(Product), nameof(Product))]
public partial class ProductPage : ContentPage
{
    public ProductPage()
    {
        InitializeComponent();
        BindingContext = this;
    }
    Product _product;
    public Product Product
    {
        get { return _product; }
        set
        {
            _product = value;
            ProductName = _product.Name;
            ManufacturerName = _product.ManufacturerName;
        }
    }
    string _productName;
    public string ProductName
    {
        get => _productName;
        set
        {
            _productName = value;
            OnPropertyChanged();
        }
    }
    ..
```

binds the parameter to a string called here, in this case, Product, on the page you are navigating to

Parameter to receive, Used *nameof* to avoid typos

Add a Product called Product to match the key in the navigation state

Call the OnPropertyChanged method to update the UI

4 Anvende dependency injection og det generiske "host builder pattern" til at registrere services og ressourcer. (MAUI)

4.1 Teori

Generic Host Builder Pattern:

- Purpose: Provides a unified way to configure and build applications in .NET, including dependency injection (DI), logging, and configuration.
- Application Initialization: In .NET MAUI, the MauiProgram.cs file uses the Generic Host Builder to initialize the application.
- Components: Includes DI, logging, configuration, and app shutdown.

Dependency Injection (DI):

- Concept: A software design pattern aimed at reducing hard-coded dependencies in an application.
- Goal: To make the design loosely coupled by removing hard-coded dependencies.
- Implementation: Achieved through an interface, allowing another class to inject dependencies at runtime.

4.2 Implementering

Generic Host Builder Pattern:

- **Application Initialization:** In the `MauiProgram.cs` file, you used the `MauiApp.CreateBuilder()` method to initialize your application.
- **Service Registration:** You registered your views, view models, and services within the `MauiProgram.cs` file using the DI container.

Dependency Injection:

Service Registration in DI-Container

```
// Register Views and ViewModels
builder.Services.AddSingleton<MainPage>();
builder.Services.AddSingleton<MainViewModel>();

builder.Services.AddSingleton<CalendarPage>();
builder.Services.AddSingleton<CalendarViewModel>();

builder.Services.AddTransient<InvoicePage>();
builder.Services.AddTransient<InvoiceViewModel>();

// Register Services
builder.Services.AddSingleton<INavigationService, NavigationService>();
builder.Services.AddSingleton<DatabaseService>();
```

- Singleton: Single instance, lifetime, Transient: New instance for each request

Constructor Injection

```
0 references
public MainViewModel(INavigationService navigationService, DatabaseService databaseService)
{
    _navigationService = navigationService;
    _databaseService = databaseService;
    Date = DateTime.Now; // Default date to today
    Time = DateTime.Now.TimeOfDay; // Default time to now
}
```

- DatabaseService skulle have været en interface for mere flexibility.
- Benefits: Decoupling og mock dependencies for test

5 Kunne anvende MVVM arkitekturen til at implementere applikationer med en grafisk brugergrænseflade ved brug af programmeringssprogene C# og XAML. (MAUI)

- Model
  Forretningslogik: Regler for at processere bruger input eller kommunikation med en API eller database
- View
  UI logik: layout og tekst formatering
- ViewModel
  Præsentation logik: UI state såsom værdier af properties og logik for at svare tilbage til brugerinteraktioner
- Bruges til separation af ansvar via bl.a. data binding
- Dermed kan ViewModel blive testet uden at have UI klar
- View er forbundet til ViewModel gennem data binding og sender kommandoer til Viewmodel (ViewModel kender ikke til View)
- ViewModel kan interagere med Model gennem properties, metodekald og kan modtage events fra model (Model kender ikke til ViewModel)
- Hvis det ikke var for ObservableObject, skulle der implementeres INotifyPropertyChanged (En event handler der fortæller UI'en når en property i en ViewModel har ændret sig)

6 Redegøre for arkitekturen for en Webapplikation. (react)

- MVC
- Client-Server model and http protocol
  Der er en server som afleverer kode til en webbrowser, og så skal man også typisk hente data et sted fra via http protocol.
  Db.json er mock server, som vi intergaerer med genenem vores api.js
- Browser engine and reacts virtual dom
  Hver browser har deres egen engine, som er en slags compiler for browseren.
  React har sin egen virtuel DOM, hvor den holder styr på UI opdateringer.

  Your use of React components leverages the virtual DOM. When state changes in components like `BookAppointmentComponent` and `EditAppointmentComponent`, React efficiently updates the DOM by re-rendering only the parts that have changed.

- Effektivitet af virtual DOM in SPA
  For ikke at spilde ressourcer, så er der en tidligere virtuel DOM og en ny virtuel DOM. React bruger hukommelsen af den tidligere DOM til at sørge for at den kun opdaterer den nye del fra den virtuelle DOM. Dermed har man en SPA, hvor konceptet er at der er én page som dynamisk opdateres med data, fremfor at der loades hele nye pages.

  Vises igen i vores components
- Hver node i en DOM har interne states og properties
  In the DOM, elements have attributes (like class, id, etc.) and can hold data. React components manage these states and properties internally, allowing for modular and maintainable code.

  Each React component manages its own state using hooks like `useState`. For example, the form inputs in `BookAppointmentComponent` have state variables that store their current values, and changes to these values trigger updates in the component.

7 Designe og implementere Webapplikationer med en grafisk brugergrænseflade med brug af HTML5, CSS og Javascript eller Typescript. (react)

- Design med HTML5 og JSX:
  Your React components are built using JSX, which combines JavaScript and HTML-like syntax to define the structure of your UI. For example, the `BookAppointmentComponent` defines a form with various input fields using JSX.
- CSS for styling
- Javescript for eventhandlers + api-kald

8 Anvende et klient side bibliotek til udvikling af Webapplikationer.

React må ikke selv gå i en DOM og lave ændringer. Dermed har man hooks som gør at en komponent kan håndtere states og sideeffekter. En sideeffekt er noget der har en effekt udenfor selve komponenten (Fetch data, DOM manipulation eller subscribe/unsubscribe til services)

- useState: Tilføj en state til en funktionel komponent, samt at gøre det muligt at ændre på den state
- useEffect: Håndtering af sideeffekter, såsom at hente data eller DOM manipulation, efter komponenten er blevet rendered.
- useContext: Gør det muligt at passere data gennem komponenttræet uden at skulle passere props manuelt gennem hvert niveau, hvilket gør dataet globalt
- useLocation: Returnere nuværende lokation objekt. Lokation objektet hvor appen er nu

9 Anvende komponenter til opbygning af en Webapplikation. (react)

Bruger funktionel komponent siden UI og logik ikke samarbejder 100% med en klasse komponent.

Tidligere brugte man class components:

> These were used earlier in React's history, combining UI and logic within a single class. This sometimes led to more complex and harder-to-maintain code, especially when dealing with state and lifecycle methods.

10 Anvende client-side routening i en Webapplikation. (react)

At React kan navigere til en anden side uden at tilgå serveren med brug af en navbar-

Server-side routing vs client side:

- 1. **Client-Side vs. Server-Side Routing**:
  - **Server-Side Routing**: Each navigation request results in a full page reload, with the server sending the entire HTML document for each new route. This can be slower and less efficient.
  - **Client-Side Routing**: Only the necessary components are re-rendered, with the JavaScript managing the URL changes without requiring a full page reload. This results in a faster and smoother user experience.

React Router:

- 2. **React Router**:
  - **React Router** is a standard library for routing in React. It enables the navigation among views of various components in a React application, allows changing the browser URL, and keeps the UI in sync with the URL.
  - **Components in React Router**:
    - `BrowserRouter`: A router implementation that uses the HTML5 history API to keep your UI in sync with the URL.
    - `Routes`: A container for all the route definitions.
    - `Route`: Defines a mapping from a URL path to a React component.
    - `Link`: A component that allows navigation to different routes defined in your application without refreshing the page.