Rising Star Pre-assignment

Tämän projektin tarkoituksena oli kehittää WPF (Windows Presentation Foundation) pohjainen sovellus, joka hakee Bitcoin historia tietoa CoinGecko-rajapinnan kautta käyttäjän määrittelemän päivämäärävälin perusteella ja näyttää tiedot graafisessa muodossa. Tehtävänannon mukaan sovellus oli tarkoitus tehdä MVC (Model-View-Controller) mallin mukaisesti, mutta tässä sovelluksessa käytin MVVM (Model-View-ViewModel) mallia koska se on suunniteltu erityisesti WPF-sovellusten ja muiden XAML-pohjaisten sovelluksien kanssa toimivaksi, mahdollistaen paremman tietojen ja käyttöliittymän erottelun, datan sitomisen sekä kehityksen modilaarisuuden.

Projektin vaatimukset

- Käyttäjän pitää pystyä määrittämään alkamis- ja päättymispäivämäärät ja ohjelma hakee Bitcoinin hinnat siltä aikaväliltä.
- Jonkinlainen graafinen kaavio, joka näyttää tiedot aikaväliltä.
- Selkeä käyttöliittymä ja helppokäyttöisyys.
- MVVM-mallin mukainen rakenne.

Sovelluksen suunnittelu

Sovellus on rakennettu MVVM-mallin mukaisesti ja sen kolme pääkomponenttia ovat:

- **Model (Malli)**: Vastaa datan käsittelystä ja tallennuksesta. Tässä projektissa Malli-luokka hakee ja käsittelee Bitcoinin hintadataa CoinGecko-rajapinnasta. Malli on erillinen käyttöliittymästä, joten se voi toimia ilman suoraa sidosta näkymään (View).
- **View (Näkymä)**: Käyttöliittymä, joka on rakennettu XAML-kielellä. Näkymä esittää käyttäjälle kaikki tarvittavat komponentit, kuten päivämäärävalitsimet, ”Fetch Data” -painikkeen ja kaavion. Näkymä ei sisällä mitään logiikkaa, vaan ainoastaan määrittää mitä sovellus näyttää ja miten käyttäjä voi vuorovaikuttaa sen kanssa.
- **ViewModel (Näkymämalli)**: Näkymämalli yhdistää Mallin ja Näkymän. Se hallinnoi logiikkaa, joka reagoi käyttäjän toimiin ja päivittää näkymän vastaamaan datan muutoksia. Näkymämalli sisältää komennot ja palveluiden käytön, jotka aktivoituvat, kun käyttäjä tekee toimia käyttöliittymässä.

Sovelluksessa on lisäksi muita komponentteja esim. komennot (Commands), palvelut (Services) ja apuluokat (Utils), jotka auttavat hallitsemaan sovelluksen vuorovaikutuksia ja käyttöliittymän monimutkaisempia toimintoja:

- **Commands (Komennot)**: Kaikki käyttäjän vuorovaikutukset tapahtuvat komentojen kautta. MVVM-mallissa komennot ovat olennainen osa logiikan ja käyttöliittymän

erottelua. Komennot toteutetaan käyttämällä ICommand-rajapintaa, joka mahdollistaa eri käyttäjätoimintojen, kuten painikkeiden klikkausten tai hiiren liikkeiden hallinnan. Tässä on esimerkki komennosta FetchDataCommand, joka suoritetaan, kun käyttäjä painaa ”Fetch Data” painiketta. Komento kutsuu näkymämallin logiikkaa, joka vuorostaan hakee tiedot mallilta ja päivittää ne kaavioon:

```
2 references
public class FetchDataCommand : BaseCommand
{
    private readonly Func<Task> fetchDataAction;

    1 reference
    public FetchDataCommand(Func<Task> fetchDataAction)
    {
        this.fetchDataAction = fetchDataAction;
    }

    1 reference
    public override async void Execute(object? parameter)
    {
        if(fetchDataAction != null)
        {
            await fetchDataAction();
        }
    }
}
```

- **Services (Palvelut)**: Sovelluksessa on eri palveluita, jotka käsittelevät tiettyjä tehtäviä, kuten kaavion piirtämistä ja vuorovaikutuksia kaavion kanssa. Palvelut mahdollistavat sovelluksen logiikan erottamisen käyttöliittymästä ja parantavat sovelluksen laajennettavuutta ja ylläpidettävyyttä. Tässä on esimerkki sovelluksen ChartService palvelusta, joka vastaa kaavion piirtämisestä. ChartService on erillinen näkymämallista, jotta kaavion logiikka pysyy erillään käyttöliittymälogiikasta:

```
2 references
public class ChartService
{
    1 reference
    public void DrawPriceChart(Canvas chartCanvas, List<Tuple<DateTime, double>> bitcoinPrices, List<Ellipse> dataPoints, List<double> dataPointPositions)
    {
        //Logiikka kaavion piirtämiseen
    }
}
```

- **Utils (Apuluokat)**: Apuluokkien tarkoituksena on yksinkertaistaa tiettyjä teknisiä haasteita. Esimerkiksi WPF-sovelluksessa tietyt käyttäjätoiminnot, kuten hiiren liikkeiden seuraaminen kaavion päällä käyttämällä sitomista (Binding) ei ole suoraan tuettu. Tässä on esimerkki MouseEvents luokan GetMouseMoveCommand funktiosta, joka mahdollistaa MouseMove-tapahtuman sitomisen näkymämalliin:

```
1 reference
public static ICommand GetMouseMoveCommand(UIElement element)
{
    return(ICommand)element.GetValue(MouseMoveCommandProperty);
}
```

Käyttöliittymän suunnittelu (View)

Sovellus on WPF-pohjainen ja käyttöliittymä on tehty käyttämällä XAML kieltä. Sovelluksessa käyttäjillä on mahdollisuus valita päivämäärä väli valitsemalla aloituspäivämäärän (Start Date) ja lopetuspäivämäärän (End Date), jonka perusteella Bitcoin tiedot haetaan. Kaavio esittää tiedot selkeässä muodossa.

Käyttöliittymän keskeiset komponentit:

- Päivämäärävalitsimet (DatePicker)
- ”Fetch Data” -painike (Button)
- Kaavio (Chart)

Esimerkki XAML-Koodista:

```
<Label Content="Start Date:" VerticalAlignment="Center" FontFamily="Arial" FontWeight="Bold" Foreground="White"/>
<DatePicker x:Name="startDatePicker" Margin="10,0" SelectedDate="{Binding StartDate, Mode=TwoWay}"/>
<Label Content="End Date:" VerticalAlignment="Center" FontFamily="Arial" FontWeight="Bold" Foreground="White"/>
<DatePicker x:Name="endDatePicker" Margin="10,0" SelectedDate="{Binding EndDate, Mode=TwoWay}"/>
```

Tietojen käsittely (Model)

Malli-luokat ovat vastuussa datan käsittelemisestä ja tallentamisesta. Tässä on esimerkki BitcoinPrice-mallista, joka hakee datan CoinGecko-rajapinnasta päivämäärävälin perusteella:

```
1 reference
public async Task FetchBitcoinDataAsync(DateTime startDate, DateTime endDate)
{
    long fromUnix = DateTimeToUnixTimestamp(startDate);
    long toUnix = DateTimeToUnixTimestamp(endDate.AddHours(1));
    string url = $"https://api.coingecko.com/api/v3/coins/bitcoin/market_chart/range?vs_currency=eur&from={fromUnix}&to={toUnix}";
    using (HttpClient client = new HttpClient())
    {
        var response = await client.GetStringAsync(url);
        var marketData = JsonConvert.DeserializeObject<MarketData>(response);
        ProcessMarketData(marketData);
    }
}
```

Liiketoimintalogiikka (ViewModel)

Näkymämalli sisältää logiikan, joka ohjaa käyttöliittymän ja Malli-luokan välistä vuorovaikutusta. Näkymämalli-luokassa on komennot, jotka kutsutaan, kun käyttäjä painaa ”Fetch Data” -painiketta. Komennon suorituksen jälkeen data ladataan ja visualisoidaan kaaviona. Esimerkki ViewModel-luokasta:

```
3 references
public class BaseViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    6 references
    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Käyttöliittymän toiminta ja kaavioiden esittäminen

Kun tiedot on haettu, ne esitetään kaaviona. Kaavio on tärkeä osa sovellusta ja sen toteuttamiseen käytettiin WPF:n Canvas-elementtiä. Käytin tätä tapaa, koska se tarjoaa täyden kontrollin kaavion ulkoasusta ja käyttäytymisestä, eikä vaatinut mitään ylimääräisten kirjastojen käyttämistä. Kaavio sisältää päivämäärän akselit, hinta-arvoja yhdistävän viivan, sekä yksittäiset datapisteet. Käyttäjä voi tarkastella kaaviosta Bitcoinin hintakehitystä määritellyltä aikaväliltä.

Yhteenveto

Tässä projektissa kehitettiin WPF-sovellus, joka hakee Bitcoinin hintatietoja CoinGecko-rajapinnasta ja esittää ne kaaviona. MVVM-mallin avulla sovelluksen rakenne pysyi selkeänä, ja se mahdollisti helpon laajennettavuuden.