

# Vareskanning



Produktrapport

# Mini-svendeprøve

**TECH**COLLEGE

**Elev:**

Omidreza Ahanginashroudkoli

**Skolepraktik:**

Teglværket 9400 Nørresundby

**Projektnavn:**

Vareskanning

**Uddannelse:**

Datatekniker med speciale i  
programmering

**Projektperiode:**

19/Sep/2022 – 25/Nov/2022

**Afleveringsdato:**

21/nov/2022

**Fremlæggelsesdato:**

25/Nov/2022

**Vejledere:**

Lærke Brandhøj Kristensen

**Elev underskrift**

**Vejleder underskrift**

---

---

# Indholds fortegnelse

Mini-svendeprøve .....	i
Læsevejledning .....	1
Indledning .....	1
Kravspecifikation .....	2
Information om teknologier .....	4
Database .....	4
Entity Framework.....	4
Web API.....	5
JSON .....	5
Mikrocontroller .....	5
NodeMCU (ESP8266) .....	5
Raspberry Pi .....	6
MAUI .....	6
Overordnet Arkitektur .....	7
Mikrocontroller .....	8
NodeMCU (ESP8266) .....	8
Web .....	12
Maui .....	14
Brugervejledning.....	21

# Læsevejledning

Denne rapport er en af to der hører til projektet Vareskanning, projekterne og rapporter blev lavet af Omidreza Ahanginashroudkoli. Der findes to slags rapporter til dette projekt. Dette er procesrapporten og den anden er produktrapporten.

Produktrapporten indeholder projektets produkt og procesrapporten indeholder forløbet af, hvordan produktet blev lavet.

Jeg vil personligt starte med at læse produktrapporten første og anbefaler, at man starter med den og får en forståelse af projektet og hvad den går ud på og bagefter kan man læse procesrapporten, hvor jeg beskriver, hvordan jeg kom frem til mit færdigt produkt.

## Indledning

Denne rapport omhandler en proof of concept løsning på en vareskanning, som beskrevet i den tilhørende procesrapport case beskrivelse.

Med brug af strekkodescanner, som er forbundet til en Raspberry Pi via USB, er vi i stand til at scanne produkter og sende dem videre til en NodeMCU, hvor vi kan se strekkoden på et oled display, bagefter bliver data sendt videre til et ASP.NET core API. API'et logger information i en MS-SQL-database.

Imidlertid kan en Cross-platform Mobile App (Maui) spørge API'et om denne information og derved give brugeren mulighed for at se de skannede produkter på deres mobiltelefon.

I denne rapport beskrives overordnede arkitektur i detaljer og hver teknologi beskrives også kort for sig. Det beskrives også, hvordan man kan sætte projektet op på en ny computer og hvordan det benyttes af brugeren.

Projektet er udviklet af Omidreza Ahanginashroudkoli, i perioden fra 19. september 2022 til 23. november 2022 med vejledning fra Lærke Brandhøj Kristensen.

# Kravsifikation

Produktet vil bestå af en Raspberry Pi, der skanner produkter og sender det til en NodeMCU, hvor den bliver vist på en oled display og sendes videre til et API, som logger det i en database. De skannede produkter vil derefter kunne aflæses på en Cross-platform Mobile App.

## Kravsifikation

I denne tabel kan man se de krav som sættes til projektets funktionalitet.

Krav ID	Kategori	Type	Krav	Prioritet
WO1	Web API	Opsætning	API skal kunne gemme data i databasen med dato og varerne.	
WF1	Web API	Funktionalitet	API skal kunne modtage data fra mikrokontroller.	
WF2	Web API	Funktionalitet	API skal have CRUD endpoints	
DO1	Database	Opsætning	Database skal kunne kommunikere med API, og skal kunne også gemme data.	
DO2	Database	Opsætning	Lav tabeller ud fra library model ved hjælp af Entity Framework Core.	
EO1	Embedded	Opsætning	Kontrolleren skal sættes op med: <ul style="list-style-type: none"><li>• NodeMCU</li><li>• Breadboard</li><li>• Skanner</li><li>• Raspberry Pi</li><li>• Oled Display</li><li>• </li></ul>	
EF1	Embedded	Funktionalitet	Mikrokontrolleren skal kunne skanne varerne.	
EF2	Embedded	Funktionalitet	Mikrokontrolleren skal kunne sende data til API 'et.	
CF1	Cross-platform App	Funktionalitet	Appen skal kunne lave data request fra API.	
CF2	Cross-platform App	Funktionalitet	Appen skal kunne skanne med sit kamera.	
CF3	Cross-platform App	Funktionalitet	Appen skal kunne opdatere sin data gennem en Pull to Refresh.	
CG1	Cross-platform App	GUI	Appen skal kunne vise data til brugerne.	
CG2	Cross-platform App	GUI	Appen kunne vise data på en Graph.	
CG3	Cross-platform App	GUI	Appen kan have theme mode	

Prioritet Niveau	Høj	Medium	Lav

## Acceptanstest

I denne tabel man kan se de Acceptanstest, jeg vil sikre at ovenstående krav er opfyldt.

Krav ID	Testbeskrivelse	Status
WO1	Er API Konfigureret korrekt i forhold til databasen?	Opfyldt
WF1	Kan API kommunikere med mikrokontroller?	Opfyldt
WF2	Kan endpoints opfyld CRUD- operation?	Opfyldt
DO1	Kan databasen kommunikere med API og opfyld de krav?	Opfyldt
DO2	Er tabeller blev lavet på korrekt måde?	Opfyldt
EO1	Er alle de nødvendige komponenter tilgængelige?	Opfyldt
EF1	Kan mikrokontrolleren skanne?	Opfyldt
EF2	Kan mikrokontrolleren kommunikere med API'et?	Opfyldt
CF1	Kan app modtage data fra API?	Opfyldt
CF2	Er hardware konfigureret korrekt?	Ikke opfyldt
CF3	Virker Pull to Refresh som det skal?	Opfyldt
CG1	kan appen præsentere oplysninger til brugerne?	Opfyldt
CG2	kan appen præsentere korrekt Graph til brugerne?	Ikke opfyldt
CG3	Kan man skifte theme mode?	Ikke opfyldt

# Information om teknologier

I denne afsnit findes information om de teknologier produktet benytter sig af.

## Database

Projektet benytter sig af en MS-SQL-database. MS-SQL Server er et relationelt databasesystem udviklet af Microsoft og udgivet i 1989.

Relationelle databaser er en type database, der giver brugerne adgang til data der er gemt i forskellige tabeller forbundet med et unikt ID eller KEY. Ved at bruge denne KEY kan brugere låse op for data entries relateret til denne KEY på en anden tabel for at hjælpe med management eller andet. I relational database management systems (RDBMS) kan brugere indtaste SQL queries for at hente de nødvendige data.

Fordelen ved en relationel database er evnen til at forbinde data fra forskellige tabeller for at skabe nyttig information. En relationel database er nem at bruge og de unikke ID'er hjælper med at eliminere duplikerende oplysninger.

Der findes 3 forskellige typer relationer i databasen. En til en relation bruges til at skabe en relation mellem to tabeller, hvor en enkelt række i den første tabel kan kun relateres til en i en anden tabel. En til mange relation bruges til at skabe en relation mellem to tabeller, hvor enkelt række i den første tabel kan relateres til en eller flere rækker i den anden tabel, men rækkerne af anden tabel kan kun relateres til den eneste række i den første tabel.

Mange til mange relation bruges til at skabe en relation mellem to tabeller, hvor hver post i den første tabel kan relatere til alle poster eller ingen poster i den anden tabel.

## Entity Framework

Projektets database er oprettet gennem Library Model via Entity Framework Core, som automatiserer mange ting i forhold til at kommunikere med databasen. Her har man benyttet code-first princippet og forklaret, hvordan databasen og relationer skulle se ud.

Entity Framework Core er udviklet af Microsoft. Entity Framework (EF) Core er en letvægts, udvidelsesbar, open source og cross-platform version af den populære Entity Framework dataadgangsteknologi. EF Core kan fungere som en object-relational mapper (ORM), som gør det muligt for .NET-udviklere at arbejde med en database ved hjælp af .NET-objects, eliminerer behovet for det meste af den dataadgangskode, der typisk skal skrives.

## Web API

Projektet benytter sig af et API som lavet i ASP.net Core.

API står for Application Programming Interface, API'er er mekanismer, der gør det muligt for to softwarekomponenter at kommunikere med hinanden ved hjælp af et sæt definitioner og protokoller. I denne projekt bliver en database lagt til at gemme data fra mikrocontroller og Maui. Der er mange fordele, når man bruger et API. Når man bruger en API som er administreret af computere, kræves der mindre menneskelig indsats og arbejdsgange kan nemt opdateres for at blive hurtigere og mere produktive. Hvis API udføres korrekt, vil det også give systemet et sikkerhedslag.

## JSON

JSON står for JavaScript Object Notation, JSON er et letvægtformat til lagring og transport af data som objekter, dataudveksling-format der er læseligt for mennesker. Det er en god måde at sende og gemme data på, fordi det er let at forstå for mennesker og derfor kan man ændre på data, hvis der er behov for.

## Mikrocontroller

En mikrocontroller er en lille computer på en enkeltstående chip. En mikrocontroller har en til flere CPU'er, RAM og programmerbare input eller output interfaces.

Projektet benytter sig af to mikrocontroller som er en NodeMCU (ESP8266) og en Raspberry Pi. Raspberry Pi skanner produkter og NodeMCU sender dem til API'et.

### NodeMCU (ESP8266)

ESP8266 er billig Wi-Fi mikrochip med indbygget TCP/IP-netværk software og mikrocontroller kapacitet. ESP8266 er udviklet af Espressif Systems og kom først på markedet i 2014. Fordele ved at bruge den er, at den er billig og nem at komme i gang med, hvis man har lidt erfaring i Arduino. Man kan lave mange små enheder, der skal kommunikeres på nettet men en lille enkelt chip.

Projektet benytter en NodeMCU som har en enkelt ESP8266 på sig. NodeMCU er en open-source IoT (internet of things) platform som udviklede i 2015. IoT er et udtryk der beskriver fysiske enheder med sensorer, processer, software og andet, som forbinder og udveksler data med andre enheder over internettet. NodeMCU giver mulighed for at upload kode fra computeren via USB, i stedet for uploade direkte til ESP8266, hvilket er mere kompliceret og har brug for special udstyr.



## Raspberry Pi

Raspberry Pi er en lille computer som integrerer CPU'en og GPU'en i et enkelt integreret kredsløb. Raspberry Pi udviklet i Storbritannien af Raspberry Pi Foundation i 2012. Raspberry Pi er en helt computer i et lille board, som kører sit eget styresystem Raspberry Pi OS, som er baseret på Linux.

Folk bruger Raspberry Pi til at lære programmeringsfærdigheder, bygge hardwareprojekter, lave hjemmeautomatisering og endda bruge dem i industrielle applikationer. Raspberry Pi giver et sæt GPIO (generelt formål input/output) pins, der giver dig mulighed for at styre elektroniske komponenter til fysisk databehandling og udforske tingenes internet (IoT).

Der bruges en Raspberry Pi 4 i projektet. Projektskoden skrevet i Python sprog i Tommy SDK.

## MAUI

.NET MAUI er en ny udgivelse fra Microsoft baseret på Xamarin Forms. MAUI står for Multi-platform App User Interface. MAUI er dog ikke en ompakket Xamarin.Forms, hele systemet er blevet opdateret for at gøre udvikling på Cross-platform endnu nemmere for udviklere. MAUI er en Cross-platform framework til at skabe native mobil og desktop apps med C# og XAML, man kan udvikle apps der kan køre på Android, IOS, macOS og Windows fra en enkelt shared code-base.

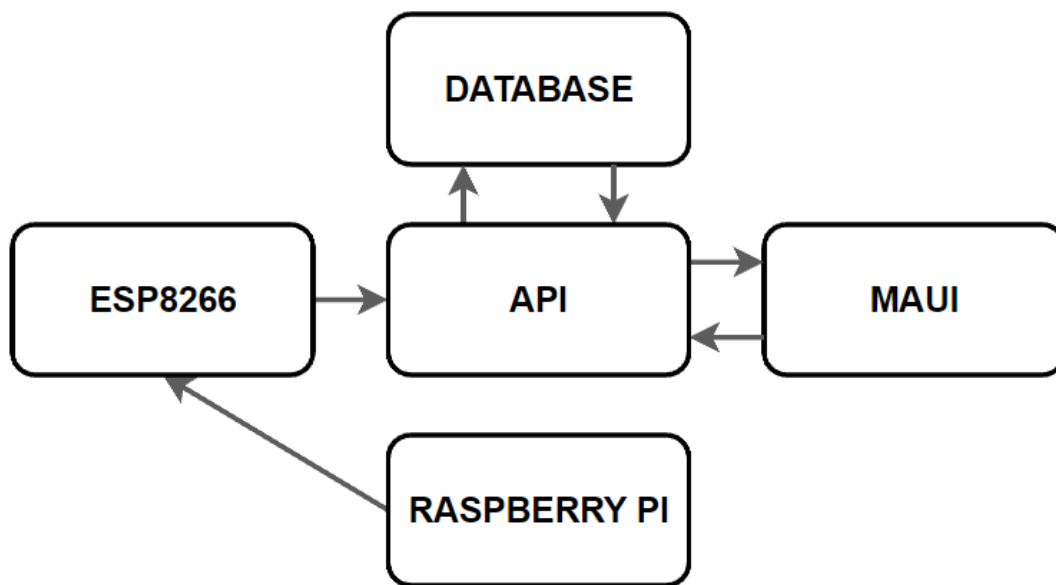
For at gøre udviklingen nemmere, .NET MAUI tilbyder en layout engine der gør det nemt at designe sidder til din app.

MAUI understøttelse af .NET's hot reload feature, der gør det muligt for apps at blive opdateret, mens de kører uden at det er nødvendigt at stoppe applikationen, den eneste ulempe er at den kun virker til front ændringer i XAML-kode.

.NET MAUI er beregnet til at være en erstatning for teknologier som WPF, UWP og Xamarin.Forms.

# Overordnet Arkitektur

Projektets overordnede arkitektur er registrering. Det betyder, at den skannede information fra en Raspberry Pi vil blive sendt via Serial kommunikation til et NodeMCU derfra ville sendes videre til et API, der gemmer det i en database. Denne information kan hentes via samme API på et Maui projekt. Denne arkitektur er afbildet nedenunder.



*Arkitekturdiagram tegnet via Draw.io*

I de følgende afsnit vil jeg uddybe og forklare hver del individuelt.

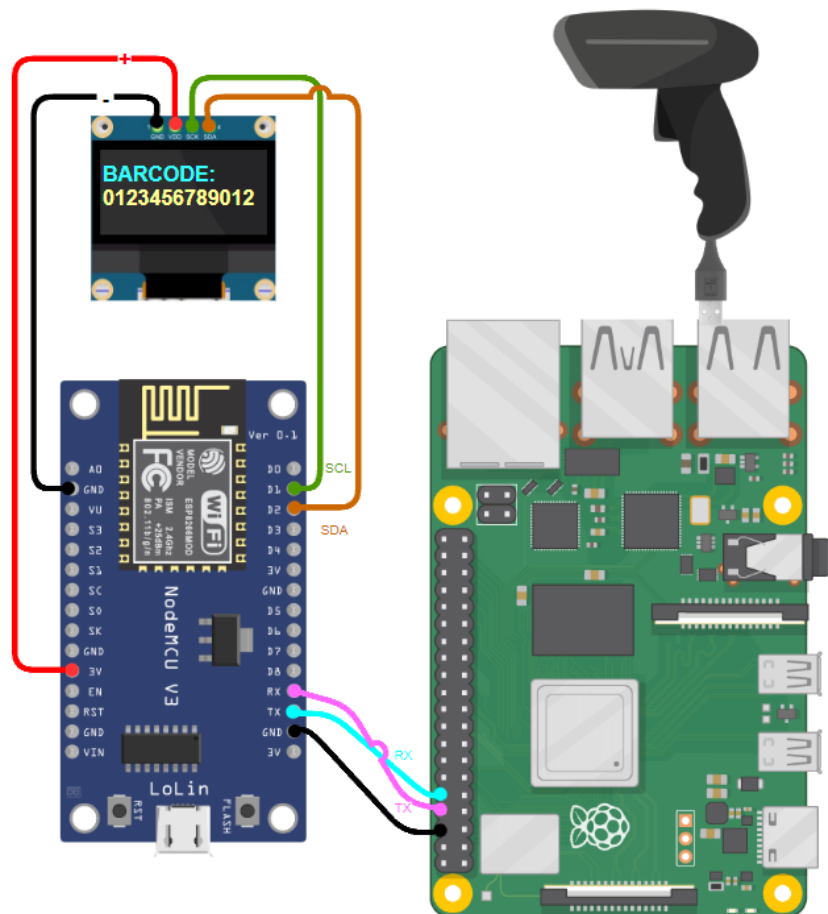
## Mikrocontroller

Projektet benytter sig af to mikrocontrollere, hvor den ene fungerer som strekkodescanner og den anden kommunikerer med API'et.

### NodeMCU (ESP8266)

ESP8266 er ifølge diagrammet forbundet til et Oled Display og en Raspberry Pi 4.

Koden til ESP8266 er skrevet i C++, i Visual Studio IDE. Nedenfor kan man se den del af koden, der viser strekkoden på en Oled Display og sender den til API'et.



*Diagrammet tegnet via Draw.io*

```
-----CODE START -----
// the setup function
void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);
    Serial.println("Connecting");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.print("Connected to WiFi network with IP Address: ");
    Serial.println(WiFi.localIP());

    // Address 0x3D for 128x64
    if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C))
    {
        Serial.println(F("SSD1306 allocation failed"));
        for (;;);
    }
    //Serial.flush();
    delay(10);
    display.clearDisplay();
}

// the loop function
void loop() {
    if (Serial.available()) {
        // Read barcode from serial
        value = Serial.readStringUntil('\n');
        Serial.println(value);
    }
    if (value.length() > 1)
    {
        // Display scanned barcode on oled
        display.clearDisplay();
        display.setCursor(0, 0);
        display.setTextSize(2);
        display.setTextColor(WHITE);
        display.println("BARCODE:");
        display.println(value);
        display.display();

        // Send POST Request
        HttpPost();

        delay(2000);
        value = "";
    }
}
-----CODE END -----
```

Denne kode kører inde i programmets `void loop()`, hvilket vil sige at det kører om og igen så længe mikrocontrolleren har strøm, efter kodens `void setup()`, som kun kører en gang. I `void setup()` sættes de ting programmet skal bruge op. For eksempel gøres Oled Display og serial terminalen klar. Så den kan printe output i konsollen eller skærmen og er forbundet til WiFi-netværket, den skal kommunikere over. Når `void setup()` er færdigt med at køre, kan koden gå ind i `void loop()` metode, hvor den kører indtil strømmen bliver taget.

```
-----CODE START -----
void HttpPost() {
    //Check WiFi connection status
    if (WiFi.status() == WL_CONNECTED) {

        // Your Domain name with URL path or IP address with path
        http.begin(client, serverName);

        // Specify content-type header
        http.addHeader("Content-Type", "application/json");

        // Data to send with HTTP POST
        String content = "{\"Barcode\":\"" + String(value) + "\", \"UserId\":\
\"0V8F2Fjo1D\"}";

        // Send HTTP POST request
        int httpCode = http.POST(content);
        // Get response
        String payload = http.getString();

        Serial.print("HTTP Response code: ");
        // Print HTTP return code
        Serial.println(httpCode);
        // Print response
        Serial.println(payload);

        //Close connection
        http.end();
    }
    else {
        Serial.println("WiFi Disconnected");
    }
}
-----CODE END -----
```

Når projektet starter med at køre, ESP8266 begynder at kontrollere igennem sine serial port om der er noget data sendt fra Raspberry Pi. De sendte data vil blive skrevet på en Oled Display at have en visuel repræsentation af skannet stregkode. Koden går videre til at kontrollere, at den stadig har ordentligt forbindelse til internettet. Hvis den ikke længere korrekt på nettet, udskriver den en fejl. Hvis den har gyldig forbindelse, vil den generere et API-kald, der indeholder stregkode, samt hvilken bruger den kommer fra i JSON format. Hvis man kigger nærmere på dette opkald, vil det fremgå at brugeren(omid0152), som informationen kommer fra er hardcoded i kaldet, da dette som nævnt kun er et proof of concept. Når kaldet er lavet, udskrives en httpkode, der viser om kaldet er gået ordentligt igennem. Kode 200 betyder, at kaldet gik igennem.

Koden til ESP8266 kan ses på GitHub link: <https://github.com/Omid2121/Mini-Svendeprove>

## Raspberry Pi

Raspberry Pi er ifølge diagrammet forbundet til en ESP8266 gennem RX og TX pins.

Koden til Raspberry Pi er skrevet i Python, via Thommy IDE. Nedenfor kan man se den del af koden der sender den skannede stregkode til ESP8266, via seriel kommunikation. Raspberry Pi er konfigureret til at starte scriptet når opstart er gennemført.

-----CODE START -----

```
1 import serial
2 import time
3 import datetime
4
5 if __name__ == '__main__':
6     #if connected via USB. Baud must be the same on NodeMCU
7     # ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)
8
9     #if connected via serial Pin(RX,TX). Baud must be the same on NodeMCU
10    ser = serial.Serial('/dev/ttyS0', 115200, timeout=1)
11    ser.flush()
12
13    while True:
14        barcode = str(input("Scan a barcode: "))
15        barcode = barcode.encode('utf-8')
16        ser = write(barcode)
17        time.sleep(0.5)
18
19
```

-----CODE END -----

Denne kode kører i et loop efter opsætningen af programmet. I koden vi fortæller Raspberry Pi at den er forbundet til en ESP8266 via RX og TX pins og skannet data skal sendes på Baud rate på 115200 således at ESP8266 har mulighed for at læse den. Stregkoden vil blive læst og kodet, derefter blive skrevet på Serial port.

Koden til Raspberry Pi kan ses på GitHub link: <https://github.com/Omid2121/Mini-Svendeprove>

## Web API

Endpoints er de adresser, du kalder for at kommunikere med en API.

Projektet API er opdelt i tre controllere, såsom Sale, User og Product. Hver har sin egen CRUD operation, der kan anmodes om fra andre systemer.

Sale		^
GET	/api/Sale	▼
POST	/api/Sale	▼
DELETE	/api/Sale/{id}	▼

Sale CRUD-operation har 3 forskellige endpoints, såsom GET, POST og DELETE.

GET endpoint returnerer en liste over alle eksisterende salg.

Når en af ovenstående koder køres, sender de information til POST endpoint i projektets API. API opretter derefter nyt salg og gemmer de givne oplysninger i databasen. DELETE endpoint er i stand til at slette brugerens salg fra databasen ved givet id.

Product		^
GET	/api/Product	▼
POST	/api/Product	▼
PUT	/api/Product/{id}	▼
DELETE	/api/Product/{id}	▼

Product CRUD-operation har 4 forskellige endpoints, såsom GET, POST, PUT og DELETE.

GET endpoint returnerer en liste over alle eksisterende produkter. POST endpoint giver brugerne mulighed for at oprette et nyt produkt, så længe den ikke eksisterer i databasen. Ved at kalde PUT endpoint er du i stand til at justere produktets oplysninger ved givet id. DELETE endpoint er i stand til at slette produktet fra databasen ved givet id.

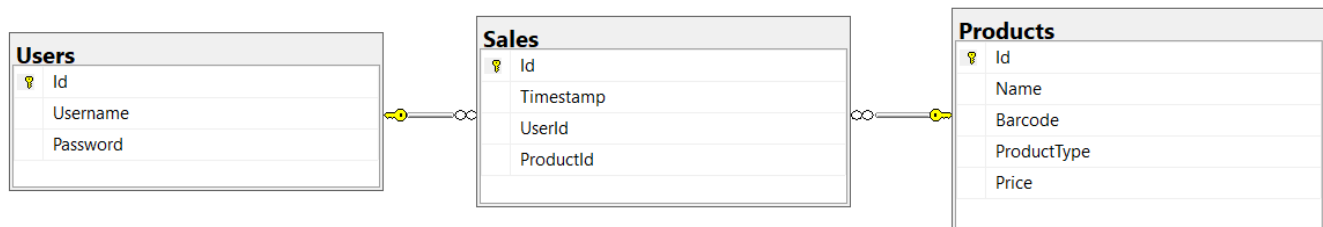
User		^
POST	/api/User/login	▼
GET	/api/User	▼
POST	/api/User	▼
GET	/api/User/{id}	▼
PUT	/api/User/{id}	▼
DELETE	/api/User/{id}	▼

User CRUD-operation har 6 forskellige endpoints, såsom Login, GET med Id, GET, POST, PUT og DELETE. Login endpoint autentikerer brugerens brugernavn og adgangskode findes i databasen. For at brugerne kan logge ind på deres konto på mobilappen. GET endpoint returnerer en liste over alle eksisterende brugere. GET med Id endpoint returnerer brugeren med givet Id. POST endpoint modtager et brugernavn og adgangskode, sørger for at de ikke eksisterer i forvejen i databasen, hvis alt er godkendt, vil det oprette en ny konto. Ved at kalde PUT endpoint er du i stand til at justere brugerens oplysninger ved givet id. DELETE endpoint er i stand til at slette brugeren fra databasen ved givet id.

Koden til API kan ses på GitHub link <https://github.com/Omid2121/Mini-Svendeprove>

## Database

Projektets database er struktureret som det fremgår af entity relation diagrammet herunder.



*Entity Relation Diagram (ERM) genereret af Microsoft SQL Server Management Studio*

Databasen har tre tabeller, Products, Users og Sales.

Products tabel har en til mange relation til Sales tabel. ProductId har en fremmerelation til Id i Products og dette betyder at et produkt kan have mange salg, men et salg kan kun have et produkt. Products tabel er en slags af et produkt, såsom en Energy drink. Products tabel bliver brugt til at forklare Sales tabel, hvilket produkt blive solgt.

Users tabel har en-til-mange relation til Sales tabel. UserId har en fremmerelation til Id i Users, og dette betyder, at en bruger har mange salg, men et salg kan kun have en bruger.



## Maui

Maui er opdelt i to via databinding jævnfør MVVM mønstret. Den visuelle side skrevet i XAML og logik del er skrevet i C#. Herunder kan man se de visuelle elementer på siden.

API kalder håndteret i MAUI Services. Der finders CRUD-operation til at kommunikere med API'et.

Siden min ProductService og SalgService er ret ens, jeg vil kun fremvise koden til mine ProductService.

```
-----CODE START -----
    public override async Task<IEnumerable<Product>> GetItemsAsync()
    {
        if (Connectivity.Current.NetworkAccess == NetworkAccess.Internet)
        {
            var products = new List<Product>();
            var client = new HttpClient();
            HttpResponseMessage response =
client.GetAsync(PRODUCT_URL).Result;

            if (response.IsSuccessStatusCode)
            {
                products = await
response.Content.ReadFromJsonAsync<List<Product>>();
            }
            return await Task.FromResult(products);
        }
        return null;
    }
----- CODE END -----
```

Ovenstående kode laver et API kald for at hente alle de eksisterende produkter i databasen. Det første, koden gør, er at sikre sig, programmet har adgang til internettet. Efter netværk er kontrolleret, ved at foretage en Client.GetAsync request, data bliver hentet i en JSON-file. Til sidst resultatet af API kalde vil vende tilbage til brugeren.

```
-----CODE START -----
    public override async Task<bool> AddItemAsync(Product product)
    {
        if (Connectivity.Current.NetworkAccess == NetworkAccess.Internet)
        {
            string json = JsonConvert.SerializeObject(product);
            StringContent content = new StringContent(json, Encoding.UTF8,
"application/json");

            HttpClient client = new HttpClient();
            HttpResponseMessage response = client.PostAsync(PRODUCT_URL,
content).Result;

            if (response.IsSuccessStatusCode)
            {
                return await Task.FromResult(true);
            }
        }
        return false;
    }
-----CODE END -----
```

Ovenstående kode laver et API kald for at oprette et ny produkt i databasen. Koden kontrollerer om der findes netværk forbindelse. Derfra de givne oplysninger fra brugeren vil sendes som en JSON-file til API'et. Til sidst resultatet af API kalde vil vende tilbage til brugeren.

```
-----CODE START -----
public override async Task<bool> DeleteItemAsync(string id)
{
    if (Connectivity.Current.NetworkAccess == NetworkAccess.Internet)
    {
        var client = new HttpClient();
        HttpResponseMessage response = await
client.DeleteAsync(PRODUCT_URL + "/" + id);

        if (response.IsSuccessStatusCode)
        {
            return await Task.FromResult(true);
        }
    }
    return false;
}
```

-----CODE END -----

Ovenstående kode laver et API kald for at slette eksisterende produkt i databasen. Koden kontrollerer om der findes netværk forbindelse. Derfra dette valgt produkt vil blive slettet fra databasen. Til sidst resultatet af API kalde vil vende tilbage til brugeren.

```
-----CODE START -----
<Grid>
    <chart:SfCartesianChart>
        <chart:SfCartesianChart.Title>
            <Label Text="Sales Chart History"/>
        </chart:SfCartesianChart.Title>

        <chart:SfCartesianChart.Legend>
            <chart:ChartLegend/>
        </chart:SfCartesianChart.Legend>

        <chart:SfCartesianChart.XAxes>
            <chart:CategoryAxis>
                <chart:CategoryAxis.Title>
                    <chart:ChartAxisTitle Text="Name"/>
                </chart:CategoryAxis.Title>
            </chart:CategoryAxis>
        </chart:SfCartesianChart.XAxes>

        <chart:SfCartesianChart.YAxes>
            <chart:NumericalAxis>
                <chart:NumericalAxis.Title>
                    <chart:ChartAxisTitle Text="Time"/>
                </chart:NumericalAxis.Title>
            </chart:NumericalAxis>
        </chart:SfCartesianChart.YAxes>

        <!--Initialize the series for chart-->
        <chart:ColumnSeries Label="Amount"
            EnableTooltip="True"
            ShowDataLabels="True"
            ItemsSource="{Binding SalesDT0}"
            XBindingPath="Name"
            YBindingPath="Timestamp">
            <chart:ColumnSeries.DataLabelSettings>
                <chart:CartesianDataLabelSettings
LabelPlacement="Inner"/>
            </chart:ColumnSeries.DataLabelSettings>
        </chart:ColumnSeries>
    </chart:SfCartesianChart>
</Grid>
```

-----CODE END -----

Denne kode skaber en kolonne graf, hvor man har mulighed for at se salghistorikken.

Grafen udfyldes og opdateres med koden nedenfor.

```
-----CODE START -----
[ ICommand ]
public async void GetSales()
{
    try
    {
        SalesDTO = (List<SaleDTO>)await saleService.GetSalesAsync();
        SalesDTO.Select(s => new SaleDTO
        {
            Timestamp = s.Timestamp,
            Name = s.Name,
        });
    }
    finally
    {
        IsBusy = false;
    }
}
-----CODE END -----
```

Denne kode kører når siden hentes.

```
-----CODE START -----
<RefreshView Command="{Binding GetProductsCommand}"
    IsRefreshing="{Binding IsBusy}"
    RefreshColor="#9003fc">

    <VerticalStackLayout>
        <Button Text="Create" TextColor="White"
            BackgroundColor="LimeGreen"
            Grid.Column="1" HorizontalOptions="End" Margin="10"
            HeightRequest="50" FontSize="15"
            Command="{Binding ContinueCommand}"/>

        <Label Text="LIBRARY" Padding="20,5,0,5"/>
        <CollectionView ItemsSource="{Binding Products}">
            <CollectionView.EmptyView>
                <Label Text="No products found" HorizontalOptions="Center"
                    VerticalOptions="Center" />
            </CollectionView.EmptyView>
            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <SwipeView>
                        <SwipeView.RightItems>
                            <SwipeItems>
                                <SwipeItem BackgroundColor="Red"
                                    IconImageSource="delete.png"
                                    Command="{Binding
                                        Source={RelativeSource AncestorType={x:Type viewModel:ProductViewModel}},
                                        Path=DeleteProductCommand}"/>
                            </SwipeItems>
                        </SwipeView.RightItems>
                        <Frame CornerRadius="5" Margin="10">
                            <Grid ColumnDefinitions="2*,2*"
                                RowDefinitions="*,*" Padding="10">
                                    <Label Text="{Binding Name}" Grid.Column="0"
                                        Grid.Row="0" FontAttributes="Bold" FontSize="Body" TextColor="Black"/>
                                    <Label Text="{Binding SelectedProductType}"
                                        Grid.Column="1" Grid.Row="0" TextColor="Black" HorizontalOptions="EndAndExpand"/>
                                    <Label Text="{Binding Barcode}"
                                        Grid.Column="0" Grid.Row="1" FontAttributes="Bold" FontSize="Body"
                                        TextColor="Black"/>
                                    <Label Text="{Binding Price}" Grid.Column="1"
                                        Grid.Row="1" FontAttributes="Bold" TextColor="Black"
                                        HorizontalOptions="EndAndExpand"/>
                                </Grid>
                            </Frame>
                        </SwipeView>
                    </DataTemplate>
                </CollectionView.ItemTemplate>
            </CollectionView>
        </VerticalStackLayout>
    </RefreshView>
-----CODE END -----
```

Produktsiden og salgssiden er ret ens. De har begge en CollectionView hvor data vil blive vist på. Der findes også en Create knap, der brugerne vil blive navigeret til en oprettelsesside. Alle elementer pakket ind i en RefreshView, der brugere kan trække for at opdatere siden.

```
-----CODE START -----
[Command]
public async void GetProducts()
{
    try
    {
        Products = (List<Product>)await productService.GetItemsAsync();
        // Doesn't work
        foreach (var product in Products)
        {
            SelectedProductType = Enum.GetName(typeof(ProductType),
product.ProductType);
        }
    }
    finally
    {
        IsBusy = false;
    }
}

[Command]
public async void AddProduct()
{
    if (!string.IsNullOrEmpty(Name) &&
!string.IsNullOrEmpty(Barcode))
    {
        await productService.AddItemAsync(new Product
        {
            Name = Name,
            Barcode = Barcode,
            ProductType = Enum.Parse<ProductType>(SelectedProductType),
            Price = Price
        });
    }
    await Shell.Current.GoToAsync($"://{nameof(ProductPage)}");
}

[Command]
public async void DeleteProduct(Product product)
{
    await productService.DeleteItemAsync(product.Id);
}

/// <summary>
/// This method is called when user presses the ContinueCommand on
ProductPage.
/// </summary>
public override async void Continue()
{
    await Shell.Current.GoToAsync(nameof(ProductCreationPage));
}
-----CODE END -----
```

ViewModel for produkter og salg er ret ens. Som koden overfor viser, er der kommandoer til at hente, tilføje og slette data. Inde i metoderne kontrollerer man om brugeren har udfyldt tekstboksene med oplysninger. Services kaldes for at sende en request til API'et.

# Opsætning

Når man sætter projektet op til at køre på en ny maskine, så er nogle justeringer nødvendige for at gøre projektet kompatibelt med det nye miljø.

Projektets API kører lokalet på maskinen. Ved hjælp af Conveyor by Keyoti extension, gives der ip-adresse som kan bruges i vores mikrocontroller og MAUI projekter.

Man kan hente projekter gennem dette Github link: <https://github.com/Omid2121/Mini-Svendeprove>

På dette billede kan man se forside som er en loginside.

For at kunne logge ind på sine konto, man skal skrive sine brugernavn og adgangskode og tryk på login-knappen.

Der findes også en knap til folk uden en konto. Her vil brugeren blive navigeret til lignende side for at oprette en ny konto.



The image shows two screenshots of the app's authentication screens. The left screenshot is the 'Sign up' screen, featuring a blue header, a blue scanner icon, and input fields for 'New username' and 'New password'. It has a blue 'SIGN UP' button and a link for 'Login'. The right screenshot is the 'Log In' screen, also with a blue header and scanner icon, but with input fields for 'Username' and 'Password'. It has a blue 'LOG IN' button and a link for 'Sign Up'.

Efter login-processen vil appen navigere brugeren til Overview siden.

Den viste side angiver til brugeren en graf, der viser en historik over de solgte produkter.

Ved at navigere til Cart siden, man kan se øverst på siden, der findes to faner en med eksisterende produkter og en anden med brugersalg.

På dette billede kan man se øverst til højre på produktsiden er der en knap til oprettelse af nyt produkt. Som vil navigere brugeren til en ny side, hvor den er i stand til at lave et nyt produkt.

Produktsiden giver brugeren en liste over eksisterende produkter. Hvert rektangel repræsenterer et produkt. Inde i rektanglet vises detaljer om hvert produkt, såsom produktets navn, stregkode, typen og prisen på den.

10:03

Product Creation

Add new product

Name

Barcode

Category

Other

Price

0

Create

9:58

Products

PRODUCTS SALES

Create

LIBRARY

Booster 5741000142899	15.00
test 123456789012	50.00
Monster Ultra 5060337502290	12.00


10:22

Products

PRODUCTS SALES

Create

LIBRARY

1142899	15.00	
test 123456789012	50.00	
Monster Ultra 5060337502290	12.00	

Overview

Cart

Scanner

Overview

Cart

Scanner

Ved at swipe til højre på et produkt vil der være mulighed for at fjerne produktet som vist på billedet.

Overview

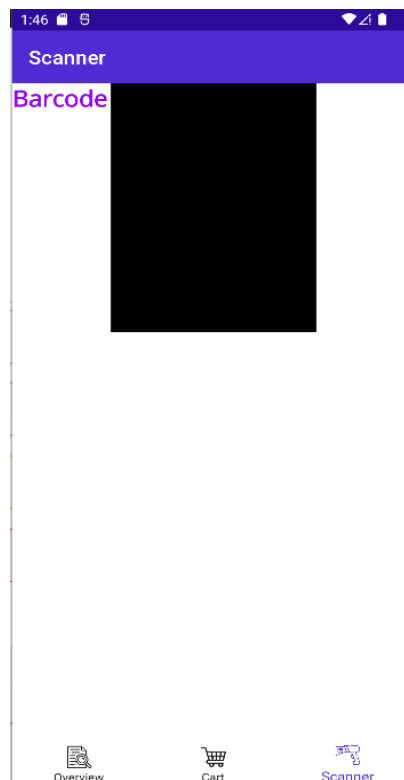
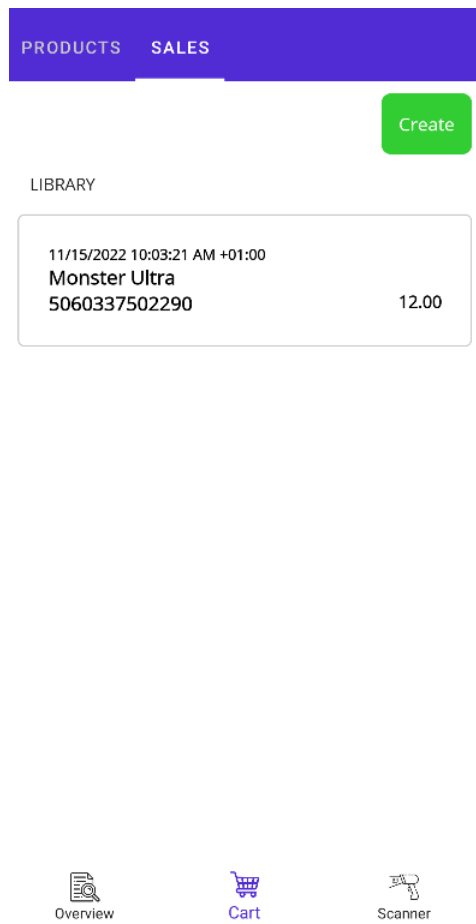
Cart

Scanner

På dette billede kan man se, en liste over brugersalg.

Man kan se øverst til højre på salgssiden er der en knap til oprettelse af nyt salg. Det vil navigere brugeren til Scanner side, hvor de er i stand til at lave et nye salg.

Salg siden giver brugeren en liste over brugerens salg. Hvert rektangel repræsenterer et salg. Inde i rektanglet vises detaljer om hvert salg, såsom salgstid, navn, stregkode, typen og prisen på det.



På denne side kan brugeren oprette nyt salg. Ved hjælp af telefonens kamera er brugeren i stand til at scanne nye vare. Denne side udfører det sammen med mikrocontrolleren.

Brugeren er i stand til at oprette nyt salg ved at få adgang til telefonens kamera.

# Brugervejledning

Projektet er en Proof Of Concept og den er ikke færdig med at udvikle. For at køre projektet skal man have alle dele af det, da projektet kører lokalt. For at kunne have forbindelse gennem hele projektet, skal APIs ip-adresse justeres. Conveyor by Keyoti giver en ip-adresse til api'et, som kan bruges i mikrocontroller til at lave POST requests. Samme angivne adresse bruges på frontend services for at gøre kommunikation muligt.