

# ShopXpress



Produktrapport



# TITELBLAD

## Elev

Omidreza Ahanginashroudkoli

## Projektnavn

ShopXpress

## Uddannelse

Datatekniker med speciale i programmering

# TECHCOLLEGE

Tech College Aalborg  
Struevej 70,  
9220 Aalborg

## Projektperiode

06.nov.2023 – 08.dec.2023

## Afleveringsdato

01.dec.2023

## Fremlæggelsesdato

07.dec.2023

## Vejledere

Frank Rosbak  
Lars Thise Pedersen

## Censor

(Navn)



# INDHOLDSFORTEGNELSE

Titelblad .....	2
Indholdsfortegnelse.....	3
Læsevejledning .....	4
OverblikDiagram .....	5
Kravspec & Testkonditioner .....	6
Overordnet arkitektur .....	9
Brugervejledninger .....	24
Konklusion .....	32
Kidelite .....	33

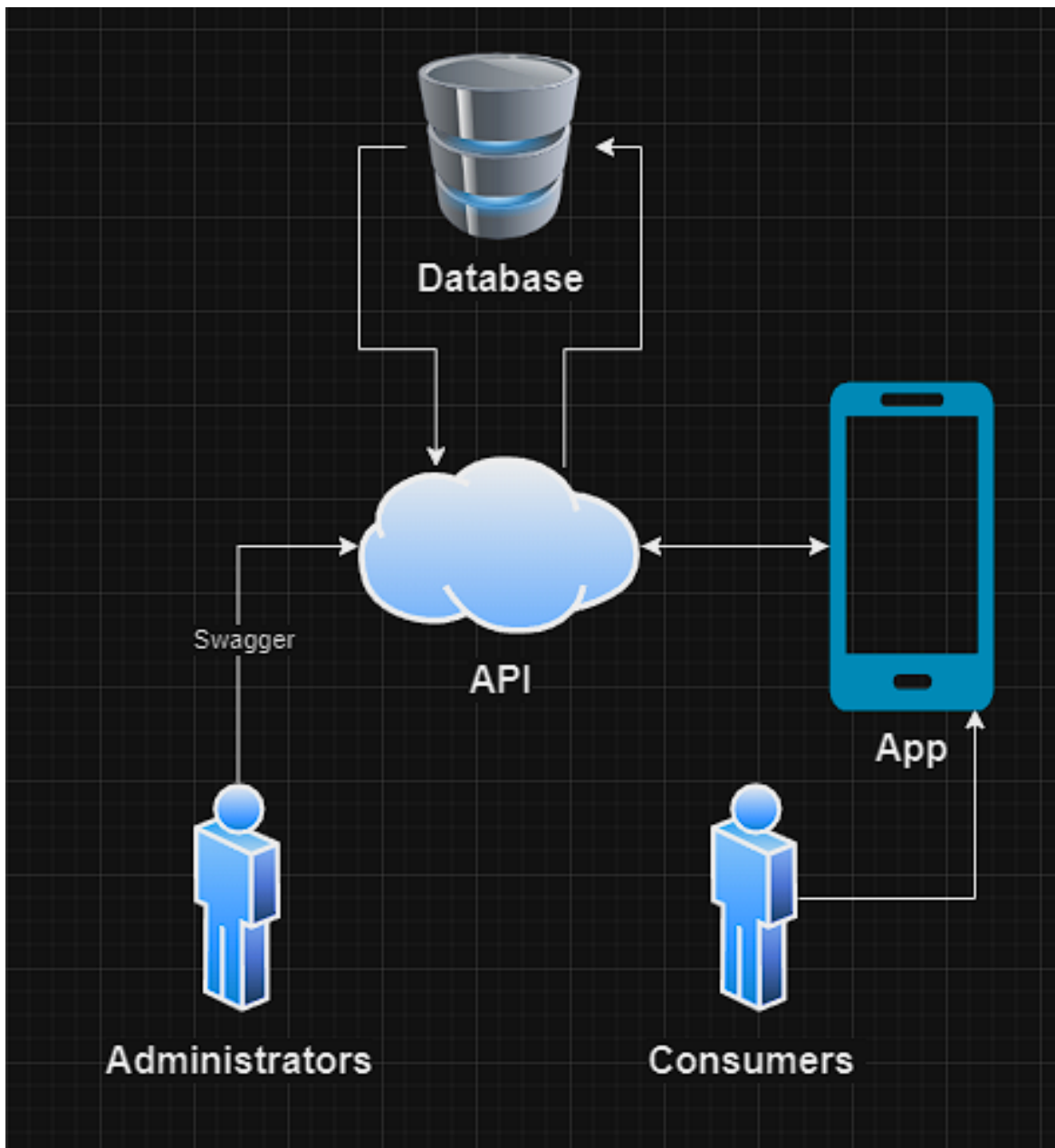


# LÆSEVEJLEDNING

Dette dokument skal læses sammen med Procesrapporten for at få det fulde overblik over produktet og processen bag skabelsen. Der vil undervejs forekomme diagrammer, figurer og tegninger, der anbefales kigge disse illustrationer, da disse kan være vigtigste for forståelsen af indholdet.

# OVERBLIKSDIAGRAM

Diagrammet herunder illustrerer et simplificeret overblik over ShopXpres's opbygning.





# KRAVSPEC & TESTKONDITIONER

I dette afsnit kan man se alle kravene og deres tilstand.

Tabellen nedenfor er kravene til brugere med enhver rolle (administratorer og forbruger).

ID	Krav	Prioritet	Beskrivelse	Opfyldt	Delvist opfyldt	Ikke opfyldt
K1	Bruger oprettelse	1	Det skal være muligt at oprette sig som bruger i systemet.	X		
K2	Bruger login og logout	1	Det skal være muligt at logge sig ind som sin egen bruger med e-mail og kodeord. Det skal være muligt at logge sig ud.	X		
K3	Hent egen bruger data	1	Det skal være muligt at hente sin egen brugerkonto, med dens persondata.	X		
K4	Redigere egen bruger data	1	Det skal være muligt at ændre sine egne persondata.	X		
K5	Slet egen bruger	1	Det skal være muligt at slette sin egen bruger konto med alt data. Efter lovgivning regler.	X		
K6	Hent de fire mest populære produkter.	3	Det skal være muligt at hente fire mest populære produkter. De fire mest solgte produkter i databasen.			X
K7	Hent et eller alle produkter	1	Det skal være muligt at hente et produkt ved dets unikke ID eller at hente alle produkterne i databasen.	X		

K8	Muligt at se en eller alle egne ordrer	1	Det skal være muligt at hente sin egen ordre ved sit unikke ID eller at hente alle egne ordrer i databasen.	X		
K9	Søg efter produkter	2	Det skal være muligt at finde produkter efter titel, beskrivelse og producent.	X		
K10	Muligt at afgive en ordre.	1	Det skal være muligt at afgive en ordre med tilhørende produkter, og angive et leveringssted, som enten kan være baseret på brugerens adresse eller nyoprettet adresse under bestillingsprocessen.	X		
K11	Muligt at se egen kurv	1	Det skal være muligt at se sin egen kurv med tilhørende varer.	X		
K12	Muligt at redigere eller slet varer fra kurven	1	Det skal være muligt at ændre eller slette varer fra sine egne kurven.	X		
K13	Muligt at rydde kurven	1	Kurven skal tømme sig selv efter den vellykkede afgivelse af ordren.	X		
K13	Muligt at se kategorier	1	Det skal være muligt at se alle kategorier der findes i databasen.	X		
K14	Muligt at filtrere efter kategori	1	Det skal være muligt at filtrere produkter efter en kategori.	X		

Tabellen nedenfor er kravene til brugere med administratorer rolle.

ID	Krav	Prioritet	Beskrivelse	Opfyldt	Delvist opfyldt	Ikke opfyldt
AK1	Produkt oprettelse	1	Det skal være muligt for administratorer at oprette et nyt produkt i databasen.	X		
AK2	Muligt at redigere eller slet produkter	1	Det skal være muligt for administratorer at ændre eller slette produkter i databasen.	X		
AK3	Muligt at se alle ordrer	1	Det skal være muligt for administratorer at hente ordrer.	X		
AK4	Tilføj administratorer	2	Det skal være muligt for nuværende administratorer at tilføje andre brugere i systemet som administratorer.			X
AK5	Muligt at se overblik	2	Det skal være muligt for administratorer at få et overblik over forskellige ting som f.eks, samlede ordremængde, forbrugerantal, leveret ordremængde og produktantal.			X
AK6	Kategori oprettelse	1	Det skal være muligt for administratorer at kunne oprette en kategori i databasen.	X		
AK7	Muligt at redigere og slet kategori	2	Det skal være muligt for administratorer at kunne ændre og slet kategorier i databasen.	X		

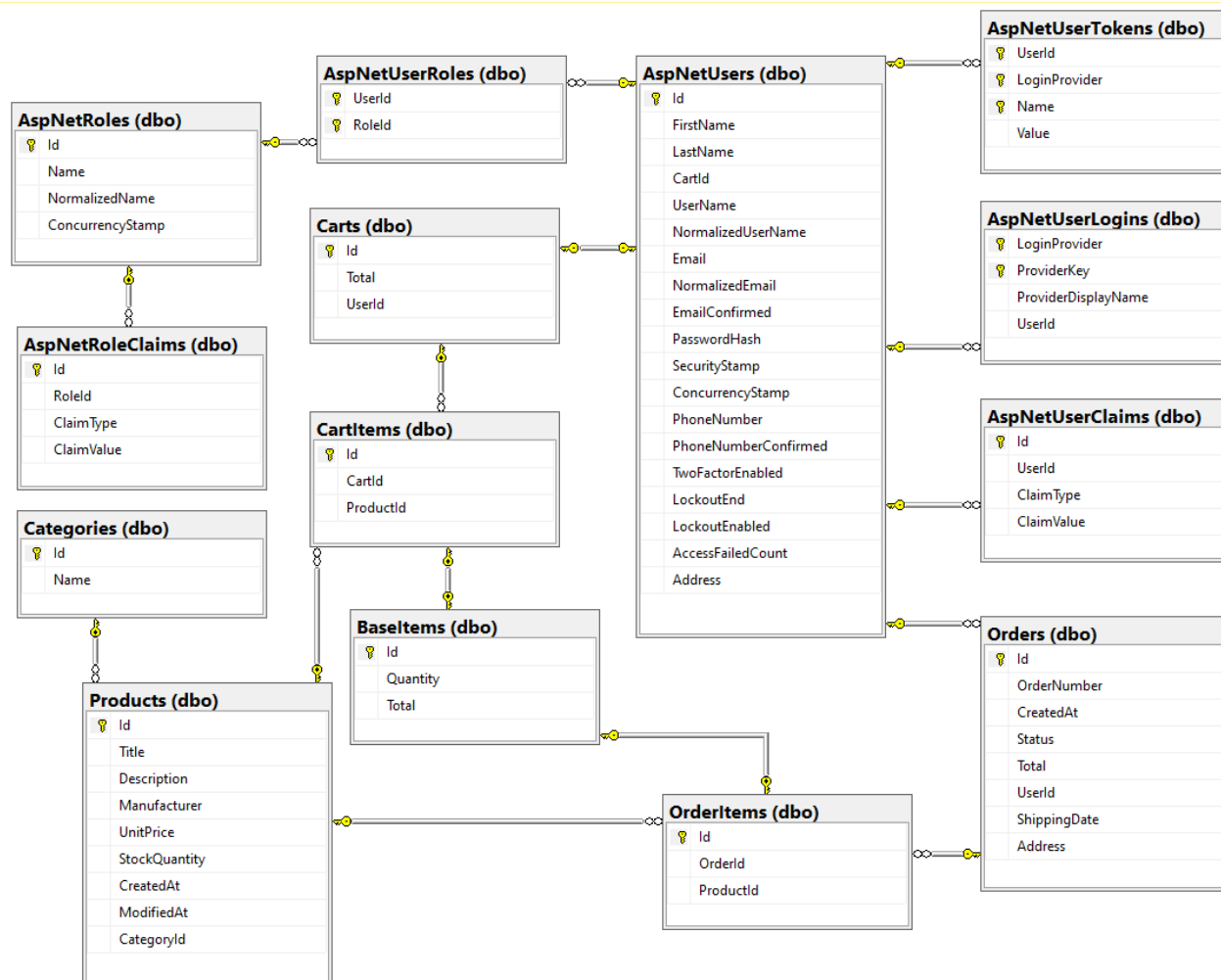


# OVERORDNET ARKITEKTUR

I de følgende afsnit vil jeg de teknologier, der anvendes i projektet og give en detaljeret fremvisning af et eksempel sammen med en forklaring.

## Database arkitektur og dokumentation

Databasen er designet med skalering, performance samt anonymitet i øje. Dette er gjort ved at opdele tabellerne samt benytte relationer på tværs af tabeller. Dette kan eksempelvis ses ved en-til-mange relationen mellem brugere og ordere, samt en TPT (Table-per-type) mapping strategi brugt i ORM (Object-relational mapping) for at repræsentere arv i en relationsdatabase. Fordelene ved TPT inheritance er, at dataintegrationen bevares. Alle tabeller i skemaet svarer til den tredje normalform. Der er ingen dataredundans.



Tabelnavn	Beskrivelse	Datatyper og felter	Evt. anmærkninger
Carts	Tabel med information om kurverne såsom det samlede ordrebeløb og hvilken bruger er til.	Id : varchar Total : decimal UserId : varchar	
Baseltems	Base tabel for OrderItems og CartItems sub tabeller med information om mængde og det samlede ordrebeløb.	Id : varchar Quantity : int Total : decimal	Baseltems er basisklassen i Tabel-per-type (TPT) mapping-strategi.
CartItems	Join table af Carts og Products, arver fra Baseltems-tabellen med antal og total af den valgte produkt.	Id : varchar CartId : varchar ProductId : varchar	CartItems er underklassen i Tabel-per-type (TPT) mapping-strategi.
Orders	Tabel med information om ordrene såsom Ordrenummer, oprettelsestidspunkt, forsendelsesdato, ordrestatus og det samlede ordrebeløb. Herunder hvilken bruger og lokation er til.	Id : varchar OrderNumber : varchar CreatedAt : DateTime ShippingDate : DateTime Status : int Total : decimal Address: varchar UserId : varchar	
OrderItems	Join table af Orders og Products, arver fra Baseltems-tabellen med antal og total af den valgte produkt.	Id : varchar OrderId : varchar ProductId : varchar	OrderItems er underklassen i Tabel-per-type (TPT) mapping-strategi.

Products	Tabel med information om produkterne såsom navn, beskrivelse og producent af produktet, pris og mængde på lager. tidspunkt for oprettelse, ændring og navnekategori.	Id : varchar Title : varchar(50) Description : varchar Manufacturer : varchar(40) UnitPrice : decimal StockQuantity : int CreatedAt : DateTime ModifiedAt : DateTime CategoryId : varchar	
Categories	Tabel med information om kategorierne.	Id : varchar Name : varchar(30)	
Users	Tabel med information om brugerne såsom fornavn, efternavn, brugernavn, e-mail, krypteret kodeord, telefonnummer og hvilken kurv er til.	Id : varchar FirstName : varchar(40) LastName : varchar(40) UserName : varchar Email : varchar PasswordHash : varchar PhoneNumber : varchar Address: varchar CartId : varchar	Adgangskoden vil blive krypteret ved hjælp af ASP.NET Core Identity

Entity Framework Core V7 er benyttet i dette projekt som ORM (Object Relational Mapping). Det er opsat efter Code First Principle, hvilket involverer, at lave modellerne i form af objekt klasser første og dernæst lade EF håndterer SQL koden. Dette sikrer, at udviklere ikke skal håndtere miljøer, for at udvikle og lave refactoring. Samtidig strømliner denne metode også udviklingen. Denne metode undgår direkte behov for adgang til databasen.

## Migrations

Migrations bliver brugt til at versionsstyre database strukturen, af samme grund som vi gerne vil versionsstyre kidekoden. Dette giver historik over databasens struktur og giver mulighed for at gå tilbage til en tidligere version.

For at oprette en ny migration har man to muligheder, den første måde er med Package Console i Visual Studio, og den anden måde at gøre det på er med CLI. Eksemplet herunder viser det med brug af PM console, med add-migration "Navn på den migration".

```
PM> Add-Migration initialCreate
```

Her kan Update-Database bruges til at tilføje alle de migrations i ConnectionStrings i "appsettings.json" filen mangler.

Som devolepment vil det ofte være den nyeste migration's ændring der bliver sendt til databasen, når projektet skal over på produktions database bliver Update-databasen kørt.

For at gå tilbage til en tidligere version kan man anvendt Update-database og valgt hvilken migration man vil tilbage til.

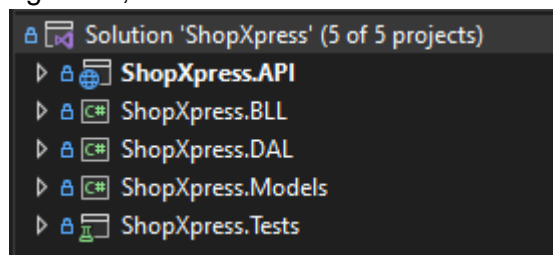
```
PM> Update-Database oldMirgration
```

## API

Serverdelen af applikationen indeholder et API, der bruges af Flutter-klientappen. Dette API håndterer alle forespørgsler fra klienten, der involverer databasen. For at sikre data og adgang er API'et sat op til gjort brug af HTTPS og autorisation af anmodninger. API'et bruger den implementerede rolleadgang for at sikre, at der kan gives adgang til de ressourcer, der efterspørges. API'et er implementeret med et Repository pattern til at adskille databaseforespørgsler fra business logic. Disse Repositories gøres tilgængelige gennem et UnitOfWork, der via. Dependency Injection kan frit bruges, hvor det er nødvendigt. For at håndtere konvertering mellem modeller og DTO'er, der bruges af klientappen, er der implementeret AutoMapper. Denne konvertering foregår i API'et, som er ansvarlig for at konvertere alle modeller, inden de udsendes, samt konvertering af DTO'er, der kommer ind via forespørgsler, før data behandles.

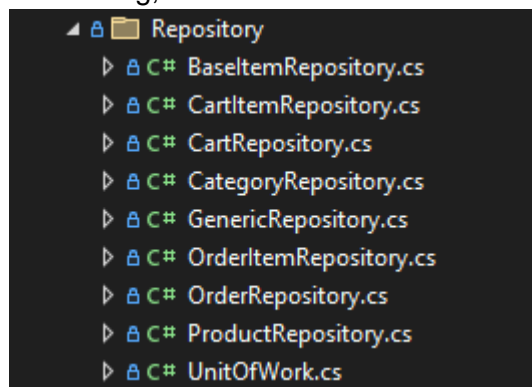
## N-layer architecture

API'et burger N-Layer design pattern til at organisere produktet i lag. Hvert lag er ansvarligt for et bestemt sæt af funktioniteter. "N" i N-Layer repræsenterer et variabelt tal, hvilket indikerer, at arkitekturen kan have et hvilket som helst antal lag afhængigt af projektets kompleksitet. Dette API er opdelt i fem lag. ShopXpress.API eller Presentation layer er det øverste lag, der interagerer med brugere og dets ansvarlighed for at præsentere informationen for brugeren og fange brugerinput. ShopXpress.BLL eller Business Logic Layer administrerer applikationens business logik og regler. Dette lag gemmer DTO-klasser, Authentication Service, CartService og Order Service. ShopXpress.DAL eller Data Access Layer styrer interaktionen med databasen og udfører operationer som lagring, hentning og opdatering af data i databasen. ShopXpress.Models eller Data er ansvarlig for opbevaring af modeller og Enums. ShopXpress.Tests-laget er ansvarlig for at teste applikationen og sikrer, at alt er som det skal være.



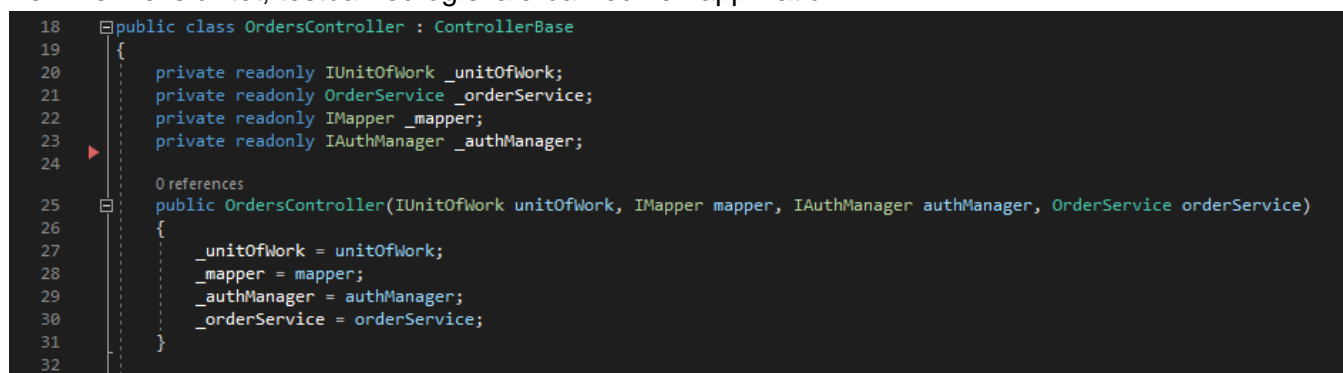
## Repository pattern og UnitOfWork

Repository pattern og UnitOfWork er patterns, der bruges i softwareudvikling for at hjælpe dataadgang og forbedre kodevedligeholdelse og testbarhed, der også fremmer best practices inden for data og transaktionsstyring. Repository pattern giver et abstrakt lag mellem applicationens business logic og dataadgangskoden. UnitOfWork er ansvarlig for at styring af transaktioner i applikationen. Det sikrer, at en række af relaterede operationer behandles som en enkelt transaktion, der enten committer alle ændringer eller ruller tilbage, hvis der opstår en fejl. Dette reducerer antallet af databasekald ved committing, når hele UnitOfWork er afsluttet. Dette forbedrer applikationens ydeevne.



## Dependency Injection

Dependency Injection er et design pattern, der bruges i softwareudvikling for at opnå en løst koblet og vedligeholdelig kodebase. Dependency Injection er en teknik hvor komponenterne i et system forsynes med deres afhængigheder i stedet for at skabe eller administrere dem internt. Denne tilgang fremmer fleksibilitet, testbarhed og skalerbarhed i en applikation.



## Transactions

Transaktioner giver mulighed for at gruppere flere databaseoperationer i en enkelt atomic UnitOfWork. Transaktioner sikrer at enten alle operationer inden for enheden er succesfuldt forpligtet til databasen, eller ingen af dem er, hvilket bevarer integriteten og konsistensen af dataene. Transaktion spiller en vigtig rolle inden for Orderservice. Når der påbegyndes at placere en ny ordre, er der behov for flere databaseoperationer for at sikre problemfri udførelse af forskellige opgaver. Ved at igangsætte en transaktion i starten af operationen og forpligte det hele til sidst, sikrer programmet at hver operation har været succesfuld. Denne tilgang øger pålideligheden og konsistensen af ordreplaceringsprocessen, hvilket reducerer sandsynligheden for delvise eller mislykkede databaseoperationer.

```
References
public IDbTransaction BeginTransaction()
{
    var transaction = _context.Database.BeginTransaction();
    return transaction.GetDbTransaction();
}

var transaction = _unitOfWork.BeginTransaction();
transaction.Commit();
transaction.Dispose();
transaction.Rollback();
```

## Entity Configuration

Modeller er konfigureret i metoden `protected override void OnModelCreating(ModelBuilder builder)`. Formålet med denne form for konfiguration er at definere, hvordan modelentiteterne skal tilknyttes den tilsvarende databasetabel, herunder detaljer såsom kolonnetyper, begrænsninger og relationer. I det følgende kodeeksempel specificerer konfigurationen detaljerne for databaseskemaet for Order entity, herunder definitionen af obligatoriske felter (Id, OrderNumber, CreatedAt, ShippingDate, Status, Total, LocationId og UserId) og deres begrænsninger (IsRequired og HasMaxLength). Konfigurationen etablerer relationer mellem Order entity og andre entities (Location, User og OrderItem). Dette informerer Entity Framework om at generere de passende foregin key relationer i databasen. Denne tilgang adskiller databasekonfigurationen fra entity klasserne, hvilket forbedrer kodeorganisering og vedligeholdelse.

```
public class OrderConfiguration : IEntityTypeConfiguration<Order>
{
    References
    public void Configure(EntityTypeBuilder<Order> builder)
    {
        // Required fields
        builder.Property(order => order.Id).IsRequired();
        builder.Property(order => order.OrderNumber).IsRequired();
        builder.Property(order => order.CreatedAt).IsRequired();
        builder.Property(order => order.ShippingDate).IsRequired();
        builder.Property(order => order.Status).IsRequired();
        builder.Property(order => order.Total).IsRequired();
        builder.Property(order => order.UserId).IsRequired();
        // Order has a one-to-many relationship with User
        builder.HasOne(order => order.User)
            .WithMany(user => user.Orders)
            .HasForeignKey(order => order.UserId);
        // Order has a one-to-many relationship with OrderItem
        builder.HasMany(order => order.OrderItems)
            .WithOne(orderItem => orderItem.Order)
            .HasForeignKey(orderItem => orderItem.OrderId);
    }
}
```

## DTO

Data Transfer Objects bruges til at overføre data mellem forskellige dele af en applikation, ofte mellem klienten og serveren. DTO'er hjælper med at forbedre effektiviteten ved kun at sende de nødvendige data frem for hele domæneobjekterne. Dette forbedrer sikkerheden, da du kan udelukke følsomme oplysninger fra at blive eksponeret for klienter. I komplekse systemer hjælper DTO'er med at nedbryde og forenkle kodens struktur og slippe af med de cirkulære afhængigheder mellem klasser. Forskellige operationer i en applikation kræver forskellige datasæt. DTO'er giver dig mulighed for at skabe

lightweight objekter, der er skræddersyet til specifikke use cases, hvilket forbedrer ydeevnen og reaktionsevnen.

```
public class CreateOrderDTO
{
    [DataType(DataType.Currency)]
    0 references
    public required decimal Total { get; set; }
    0 references
    public string? Address { get; set; }
    0 references
    public required string UserId { get; set; }
    0 references
    public virtual required ICollection<OrderItemDTO> OrderItems { get; set; }
}
4 references
public class OrderDTO : CreateOrderDTO
{
    0 references
    public required string OrderNumber { get; set; }
    0 references
    public required DateTimeOffset CreatedAt { get; set; }
    0 references
    public required DateTimeOffset ShippingDate { get; set; }
    0 references
    public required Status Status { get; set; }
    0 references
    public required Guid Id { get; set; }
    0 references
    public required UserDTO User { get; set; }
}
1 reference
public class UpdateOrderDTO : CreateOrderDTO
{
}
```

## Controllers

Controllers er ansvarlige for at modtage brugerinput, behandle det og bestemme det passende svar. Et endpoint er en specifik URL eller URI (Uniform Resource Identifier), som en HTTP request kan sendes til. Controllers hjælper med at organisere applikationen ved at gruppere relaterede handlinger sammen. For eksempel, "OrderController", der håndterer ordrerelaterede handlinger. Herunder kan man se et endpoint til at hente ordrer baseret på brugernes rolle (Administrator or Consumer). Herefter mappes der til DTO'er og returneres resultatet som et 200 OK response eller 401 Unauthorized response, afhængigt af brugernes rolle.

```
[HttpGet]
[Authorize(Roles = "Administrator, Consumer")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
0 references
public async Task<IActionResult> GetOrders()
{
    var isAdmin = User.IsInRole(Role.Administrator);
    if (isAdmin)
    {
        _orders = await _unitOfWork.Orders.GetAll();
    }
    else
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        _orders = await _unitOfWork.Orders.GetAll(o => o.UserId == currentUserId);
    }
    var results = _mapper.Map<IEnumerable<OrderDTO>>(_orders);
    return Ok(results);
}
```

Herunder kan man se et endpoint designet til at håndtere oprettelsen af ordrer via en POST request. Den udfører autorisationskontrol, validerer de indgående data, behandler ordren ved benytte af en OrderService og returnerer et passende svar med de nyoprettede ordreinformation.

```
[HttpPost]
[Authorize(Roles = "Consumer")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
0 references
public async Task<IActionResult> PlaceOrder([FromBody] CreateOrderDTO orderDTO)
{
    if (!ModelState.IsValid) return BadRequest(ModelState);
    var order = _mapper.Map<Order>(orderDTO);
    if (order.Address == null)
    {
        var user = await _authManager.GetUser(order.UserId);
        order.Address = user.Address;
    }
    var result = await _orderService.PlaceOrderAsync(order);
    if (!result) return BadRequest("Order could not be placed.");

    return Created("GetOrder", order);
}
```

## OrderService

Herunder kan man se PlaceOrder metoden, der udfører forskellige opgaver relateret til ordrebeholdning, såsom kontrol af produktlagertilgængelighed, opdatering af lagermængder, indstilling af ordredetaljer og fastholdelse af ordredata i Repositoryen. Denne metode er designet til at håndtere logikken i behandlingen af en ordre.

```
public async Task<bool> PlaceOrderAsync(Order order)
{
    // Checks if products are in stock
    await IsProductsInStock(order.OrderItems);
    var transaction = _unitOfWork.BeginTransaction();
    try
    {
        // Create order items
        var orderItems = order.OrderItems.Select(orderItem => new OrderItem
        {
            //Id = Guid.NewGuid(),
            Quantity = orderItem.Quantity,
            Total = orderItem.Total,
            ProductId = orderItem.ProductId
        }).ToList();
        // Update stock quantity
        foreach (var orderItem in orderItems)
        {
            await UpdateStockQuantity(orderItem.ProductId, orderItem.Quantity);
        }
        Order newOrder = new()
        {
            OrderNumber = Guid.NewGuid().ToString(),
            CreatedAt = DateTimeOffset.UtcNow,
            ShippingDate = DateTimeOffset.UtcNow.AddDays(3),
            Status = Status.Processing,
            Total = orderItems.Sum(item => item.Total),
            Address = order.Address,
            UserId = order.UserId,
            OrderItems = orderItems
        };
        await _unitOfWork.Orders.Insert(newOrder);
        await _unitOfWork.Save();
        transaction.Commit();
        return true;
    }
    catch (Exception)
    {
        transaction.Rollback();
        throw;
    }
}
```



```
1 reference
private async Task<bool> IsProductsInStock(IEnumerable<OrderItem> orderItems)
{
    foreach (var orderItem in orderItems)
    {
        var product = await _unitOfWork.Products.Get(p => p.Id == orderItem.ProductId);
        if (product == null || product.StockQuantity < orderItem.Quantity) return false;
    }
    return true;
}

1 reference
public async Task UpdateStockQuantity(Guid productId, int quantityChange)
{
    var product = await _unitOfWork.Products.Get(p => p.Id == productId);
    if (product != null)
    {
        product.StockQuantity -= quantityChange;

        _unitOfWork.Products.Update(product);
        await _unitOfWork.Save();
    }
    else
    {
        throw new ArgumentNullException(nameof(product), "Product not found");
    }
}
```

## Global Exception Handling

Global Exception Handling forbedrer den overordnede robusthed, vedligeholdelse og brugeroplevelse af en applikation ved at give en konsistent og kontrolleret tilgang til håndtering af fejl i hele systemet. Når der opstår en uhåndteret undtagelse, returnerer denne extension method et standardiseret JSON-formateret fejlsvare med en 500 Internal Server Error-statuskode og en fejlmeddelelse.

```
public static void ConfigureExceptionHandler(this IApplicationBuilder app)
{
    app.UseExceptionHandler(error =>
    {
        error.Run(async context =>
        {
            context.Response.StatusCode = StatusCodes.Status500InternalServerError;
            context.Response.ContentType = "application/json";
            var contextFeature = context.Features.Get<IExceptionHandlerFeature>();
            if (contextFeature != null)
            {
                await context.Response.WriteAsync(new Error
                {
                    StatusCode = context.Response.StatusCode,
                    Message = "Internal Server Error. Please Try Again Later."
                }.ToString());
            }
        });
    });
}
```

## Caching

Disse konfigurationer forbedrer tilsammen applikationens ydeevne ved at udnytte cachelmekanismen. Responses kan cachelagres i en specificeret varighed, og MustRevalidate angiver, at caches skal genvalidere det cachelagrede response med oprindelsesserveren, før det serveres til efterfølgende requests. Dette er med til at sikre, at kunderne modtager det mest opdaterede indhold.

```
public static void ConfigureCacheHeaders(this IServiceCollection services)
{
    services.AddResponseCaching();
    services.AddHttpCacheHeaders(
        (expirationOpt) =>
        {
            expirationOpt.MaxAge = 900;
            expirationOpt.CacheLocation = CacheLocation.Public;
        },
        (validationOpt) =>
        {
            validationOpt.MustRevalidate = true;
        }
    );
}
```

## Rate limiting

Rate limiting er en teknik, der bruges til at kontrollere antallet af requests fra klienter for at forhindre misbrug eller sikre rimelig brug af ressourcer. Herunder kan man se at Rate Limiting anvendes på alle endpoints med grænse på 30 request hvert 5 minut. De tilføjede services er ansvarlige for styring af Rate Limit tællere, IP policies, og den overordnede Rate Limit konfirmation.

```
public static void ConfigureRateLimiting(this IServiceCollection services)
{
    var rateLimitRules = new List<RateLimitRule>
    {
        new RateLimitRule
        {
            Endpoint = "*",
            Limit = 30,
            Period = "5m"
        }
    };
    services.Configure<IpRateLimitOptions>(option =>
    {
        option.GeneralRules = rateLimitRules;
    });
    services.AddSingleton<IRateLimitCounterStore, MemoryCacheRateLimitCounterStore>();
    services.AddSingleton<IIpPolicyStore, MemoryCacheIpPolicyStore>();
    services.AddSingleton<IRateLimitConfiguration, RateLimitConfiguration>();
    services.AddSingleton<IProcessingStrategy, AsyncKeyLockProcessingStrategy>();
}
```

## Swagger

Swagger er implementeret som et værktøj, der skal give et visuel overblik over alle API endpoints. Dette overblik giver detaljeret informationer om hvilke data et endpoint forventer og returnerer. Swagger kan også blive brugt til at teste endpoints, hvorefter test foregår gennem Postman.

## App

Flutter GetX er et kraftfuldt og let State Management pakke til Flutter. Det giver et sæt hjælpeprogrammer og klasser, der gør det nemmere at administrere tilstanden af Flutter-applikationen, håndtere afhængigheder og udføre navigation. Controllere i GetX er klasser, der er ansvarlige for at administrere tilstanden for en specifik del af applikationen. De udvider GetXController-klassen og kan bruges til at håndtere business logic, datahentning og mere.

```
class AuthenticationController extends GetxController {
```

Herunder kan man login metoden der findes i AuthenticationController. Den er ansvarlig for at håndtere brugerlogin, den validerer e-mail og adgangskode for at sikre, at de ikke er tomme. Loginoplysningerne sendes derefter i en POST request til et login endpoint. Hvis svarstatussen er 202 Accepteret, udtrækkes godkendelsestokenet og gemmes i et sikkert lokalt lager. Til sidst vil de blive navigeret til startskærmen.

```
Future<void> login() async {
  TAuthValidators.loginValidator(
    emailController.text,
    passwordController.text);

  var body = {
    'email': emailController.text,
    'password': passwordController.text,
  };
  final response = await http.post(Uri.parse('${_baseUrl}/Accounts/login'),
    headers: {'Content-type': 'application/json'},
    body: json.encode(body));
  if (response.statusCode == 202) {
    var token = json.decode(response.body)['token'];
    await storage.write(key: 'token', value: token);
    isLoggedIn.value = true;
    getUser();
    Get.offAll(() => const HomeScreen());
  }
}
```

GetX kommer med et indbygget Dependency Injection-system, der gør det nemt at administrere og indsætte afhængigheder i applikationen. Herunder kan man se at AuthorisationController bliver injiceret til til Login-siden.

```
class LoginScreen extends StatefulWidget {
  const LoginScreen({super.key});
  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final AuthenticationController authController = Get.put(AuthenticationController());
}
```

```
return Scaffold(
  body: SingleChildScrollView(
    child: Padding(
      padding: TSpacingStyle.paddingWithAppBarHeight,
      child: Column(
        children: [
          /// Logo, Title and subtitle
          const TLoginHeader(),
          Padding(
            padding: const EdgeInsets.symmetric(vertical: TSizes.spaceBtwSections),
            child: Form(child: Column(
              children: [
                /// Email
                TextFormField(...), // TextFormField
                const SizedBox(height: TSizes.spaceBtwInputFields),
                /// Password
                TextFormField(...), // TextFormField
                const SizedBox(height: TSizes.spaceBtwSections),
                /// Login button
                SizedBox(width: double.infinity, child: ElevatedButton(onPressed: () => authController.login(),
                  child: const Text("Login"))), // ElevatedButton, SizedBox
                const SizedBox(height: TSizes.spaceBtwItems),
                /// Create account button
                SizedBox(width: double.infinity, child: ElevatedButton(onPressed: () => authController.register(),
                  child: const Text("Create account"))), // ElevatedButton, SizedBox
              ],
            )), // Column, Form
          ), // Padding
    ),
```

## Sikkerhed

### HTTPS

Hypertext Transfer Protocol Secure er en sikker version af HTTP-protokollen, der bruges til kommunikation over internettet. Det bruges til at sikre overførsel af følsomme oplysninger såsom personlige data, login-oplysninger og mere. Når du opretter forbindelse til et websted med HTTPS, tjekker browseren webstedets SSL/TLS-certifikat (Secure Sockets Layer/Transport Layer Security) for at bekræfte dets autenticitet. Projektet er krypteret med TLS/SSL (HTTPS).

### ASP.NET Core Identity

ASP.NET Core Identity er et medlemskabssystem i ASP.NET Core frameworket, der forenkler autentificering og autorisation i webapplikationer. Det giver mulighed for nemt at oprette, opdatere, slette og administrere brugerkonti og understøtter forskellige godkendelsesmetoder, inklusive token-baseret godkendelse. ASP.NET Core's indbyggede autorisationssystem, der gør det nemt at kontrollere adgangen til forskellige dele af din applikation baseret på brugerroller og claims. ASP.NET Core Identity hashes automatisk adgangskoder, før de gemmer dem i databasen.

### ASP.NET Core IdentityRole

IdentityRole er en klasse, der repræsenterer en rolle i identitetssystemet. Roller er en måde at kategorisere og gruppere brugere baseret på deres tilladelser eller funktionelle ansvar.

### JWT

JWT står for Json Web Token. JWT'er bruges ofte i autentificeringsprocessen. Når en brugere logger ind, genereres en JWT på serveren og sendes tilbage til klienten. Klienten inkluderer derefter JWT i header på efterfølgende requests om at autentificere sig selv. På denne måde kan serveren stole på, at klienten der fortager request, er den oprindelige der loggede ind. Processen med oprettelse af token starter når brugeren logger ind, opretter applikationen et token med claims såsom "JWT Id, userId, e-mail, fornavn, efternavn og roller", inkluderer JWT relaterede indstillinger (issuer, audience, lifetime,

and secret key). For at forbedre sikkerheden af token er det bedst at gemme den secret key i en separat database, såsom Azure Key Vault Managed, i stedet for appsettings.json file.

## Flutter secure storage

Flutter secure storage er en Flutter-pakke, der giver et simpelt API til at gemme data sikkert på både Android og IOS enheder. Det sikrer at data er krypteret og beskyttet mod uautoriseret adgang. Det angivne token fra serveren bliver gemt i Flutter secure storage.

## Test

Microsoft.AspNetCore.Mvc.Testing med WebApplicationFactory bruges til at generere en database i Memory samt en testserver af API'et, dette giver mulighed for at lave integrationstests.

ITestOutputHelper skriver undtagelsesbeskeder under testudførelse.

```
public class WebApiApplication : WebApplicationFactory<Program>
{
    1 reference
    protected UnitOfWork UnitOfWork { get; set; }
    private readonly ITestOutputHelper _testOutputHelper;

    2 references
    public WebApiApplication(ITestOutputHelper testOutputHelper)
    {
        _testOutputHelper = testOutputHelper;
    }
}
```

CreateHost-metoden er ansvarlig for at fjerne services og tilføje nye services til testmiljøet. Såsom oprettelse af en In-memory-database, generering af et nyt tilfældigt databasenavn og erstatning af de rigtige autentificerings- og autorisations services med mock-implementering for at kontrollere og isolere autentificering og autorisationslogikken under testing.

```
protected override IHost CreateHost(IHostBuilder builder)
{
    builder.UseEnvironment("Testing");

    builder.ConfigureServices(services =>
    {
        var descriptor = services.SingleOrDefault(
            service => service.ServiceType == typeof(DbContextOptions<ShopXpressDbContext>));

        services.Remove(descriptor!);

        var authenticationDescriptor = services.SingleOrDefault(
            service => service.ServiceType == typeof(AuthenticationService));

        services.Remove(authenticationDescriptor!);

        var authorizationDescriptor = services.SingleOrDefault(
            service => service.ServiceType == typeof(IAuthorizationService));

        services.Remove(authorizationDescriptor!);

        string dbName = "InMemoryDbForTesting" + Guid.NewGuid().ToString();

        services.AddDbContext<ShopXpressDbContext>(options =>
        {
            options.UseInMemoryDatabase(dbName);
        });

        var serviceProvider = services.BuildServiceProvider();
        using (var scope = serviceProvider.CreateScope())
        using (var appContext = scope.ServiceProvider.GetRequiredService<ShopXpressDbContext>())
        {
            try
            {
                UnitOfWork = scope.ServiceProvider.GetService<IUnitOfWork>() as UnitOfWork
                ?? throw new NullReferenceException("UnitOfWork is null");
                appContext.Database.EnsureCreated();
            }
            catch (Exception ex)
            {
                _testOutputHelper.WriteLine(ex.Message);
                throw;
            }
        }
    });

    return base.CreateHost(builder);
}
```

Herunder kan man se en af testmetoden, der sigter mod at verificere at HTTP-POST request om at oprette et produkt med gyldige data returnerer en statuskode på 201 Created.

```
[Fact]
0 references
public async Task CreateProduct_WithValidModel_ReturnsCreated()
{
    // Arrange
    CreateProductDTO productDTO = new()
    {
        Title = "Test Product",
        Description = "Test Description",
        Manufacturer = "Test Manufacturer",
        UnitPrice = 10.99m,
        StockQuantity = 10,
        CategoryId = Guid.Parse("4678572e-5d7b-47c2-ab80-f8d6d8977017")
    };
    // Act
    var content = new StringContent(JsonConvert.SerializeObject(productDTO), Encoding.UTF8, "application/json");
    var response = await _client.PostAsync(HttpHelper.UrIs.Products, content);
    // Assert
    response.StatusCode.Should().Be(HttpStatusCode.Created);
}
```

## Versionsstyring

Til versionsstyring bliver teknologien Git benyttet gennem servicen Azure DevOps. Denne afhjælper også dokumentation, da for hvert commit der laves, sættes en commit besked, som beskriver hvad der er arbejdet på.




## Projektstyring

Link: [https://dev.azure.com/omid0152/ShopXpress/\\_backlogs/backlog/ShopXpress%20Team/Epics](https://dev.azure.com/omid0152/ShopXpress/_backlogs/backlog/ShopXpress%20Team/Epics)

	Order	Work Item Type	Title	State	Effort	Busin...	Value Area
+	1	Epic	Backend	Done			Business
		Feature	ShopXpress.API	Done			Business
		Feature	ShopXpress.BLL	Done			Business
		Feature	ShopXpress.DAL	Done			Business
		Feature	ShopXpress.Models	Done			Business
	2	Epic	Frontend	In Progress			Business
		Feature	Authentication	Done			Business
		Feature	Screens	New			Business
		Feature	Utils	In Progress			Business
		Feature	Models	New			Business
	3	Epic	Tests	Done			Business
		Feature	Helper	Done			Business
		Feature	Integration	Done			Business
		Feature	TestService	Done			Business
		Feature	Unit	New			Business
	4	Epic	Resources	In Progress			Business
		Feature	Reports	In Progress			Business

## Host

Microsoft Azure fungerer som hostingplatform for App Service, SQL Server og SQL database i mit projekt. Azure giver en abonnementskredit på \$100, som studerende kan aktivere. Derudover udvider Azure et gratis abonnement til hosting af App Service. Da min App Service ikke forventes at generere væsentlig trafik, valgte jeg det gratis abonnement, som det opfylder mit projekts behov. Til min database valgte jeg den mest omkostningseffektive mulighed, der er tilgængelig, som beløber sig til cirka \$ 5 om måneden.

<input type="checkbox"/> Name ↑↓	Type ↑↓	Location ↑↓	
<input type="checkbox"/>  shopxpressapi	App Service	Germany West Central	...
<input type="checkbox"/>  ShopXpressDB (shopxpressserver/ShopXpressDB)	SQL database	Germany West Central	...
<input type="checkbox"/>  shopxpressserver	SQL server	Germany West Central	...



# BRUGERVEJLEDNINGER

Denne afsnit giver sine brugere (Administrators og Consumers) en guide over installation og produkt funktionalitet.

## Swagger

### Installation

Der kræves ingen installation til Web API'et.

Følg disse trin for at bruge Web API'et:

1. Naviger til nedenstående link for at bruge applikationen som administrator.
2. bruge den medfølgende e-mail og adgangskode til at logge ind på kontoen.
3. Når du er logget ind, skal du bruge det genererede token til at autorisere kontoen.

### Anvendelse

Her kan ses guide for administratorer.

### Link og administrator konto

Link: <https://shopxpressapi.azurewebsites.net/index.html>

E-mail: [admin@gmail.com](mailto:admin@gmail.com)

Adgangskode: [Secret1234?](#)

### Log ind og autorisere på Swagger

For at logge ind som administrator i systemet, skal man åben en browser og tilgå hjemmesiden. Valg [POST api/login](#) endpoint. Indtast e-mailadressen og adgangskoden i bodyen, og klik på Execute knappen. I response afsnittet man får tokens man kan bruge til at autorisere.



[illegible]

## Get orders

For at se alle ordrer i databasen, skal man valg **GET api/orders** endpoint og klik på Execute knappen. I response afsnittet får man en liste af alle ordrer I JSON-format.

## Orders

GET

/api/Orders

^

🔒

Parameters

Cancel

No parameters

Execute

Clear

Responses

## Ændre ordrestatus

For at ændre statussen på kunders ordre, skal man vælge **PUT api/orders/{orderId}/status** endpoint, Indtast ordrens Id i feltet, og i indtast statussen i bodyen.

Status: Processing = 0, InTransit = 1, Delivered = 2.

The screenshot shows a REST client interface for a PUT request to the endpoint `/api/Orders/{orderId}/status`. The interface includes a "Parameters" section with a table for path parameters. The `orderId` parameter is marked as required and is a string, with the value `orderId` entered in the input field. Below the parameters, the "Request body" is shown as a JSON object: `{ "status": 0 }`. The media type is set to `application/json-patch+json`. A "Cancel" button is visible in the top right corner.

Name	Description
<code>orderId</code> * required <code>string(\$uuid)</code> (path)	orderId

Request body: `{ "status": 0 }`

Media type: `application/json-patch+json`

## Kategori oprettelse, redigering og slette

For at oprette en ny kategori, skal man vælge **POST `api/categories`** endpoint, indtast navn på kategorien, og klik på Execute knappen.

For at lave om på kategoriens navn, skal man vælge **PUT `api/categories/{categoryId}`** endpoint, indtast kategoriens Id i feltet, og et nyt kategorinavn i bodyen.

For at slette kategorien, skal man vælge **DELETE `api/categories/{categoryId}`** endpoint, indtast kategoriens Id i feltet, og klik på Execute knappen.

The screenshot shows a REST client interface with a list of API endpoints under the heading "Categories". The endpoints are:

- GET** `/api/Categories`
- POST** `/api/Categories`
- GET** `/api/Categories/{categoryId}`
- PUT** `/api/Categories/{categoryId}`
- DELETE** `/api/Categories/{categoryId}`

Each endpoint entry includes a method button, the endpoint path, and a lock icon.

## Product oprettelse, redigering og slette

For at oprette et ny produkt, skal man vælge **POST `api/products`** endpoint, indtast produkt oplysninger, og klik på Execute knappen.

For at lave om på et produkt, skal man vælge **PUT `api/products/{productId}`** endpoint, indtast produkt Id i feltet, og nyt oplysninger i bodyen.

For at slette produktet, skal man vælge **DELETE `api/products /{productId}`** endpoint, indtast produkt Id i feltet, og klik på Execute knappen.

Products			^
GET	/api/Products	✓	🔒
POST	/api/Products	✓	🔒
GET	/api/Products/{productId}	✓	🔒
PUT	/api/Products/{productId}	✓	🔒
DELETE	/api/Products/{productId}	✓	🔒
GET	/api/Products/search	✓	🔒

## Flutter App

### Installation

Følg disse trin for at bruge Flutter-appen:

1. Naviger til nedenstående link for at downloade Flutter APK-filen.
2. Når overførslen er færdig, skal du fortsætte med at installere appen ved at trykke på den downloadede fil.
3. Efter installationen kan du starte appen og begynde at bruge dens funktioner.
4. Sørg for at du har en aktiv internetforbindelse.

### Anvendelse

Her kan ses guide for brugere af systemet.

### Link og Consumer konto

Link: link til flutter app

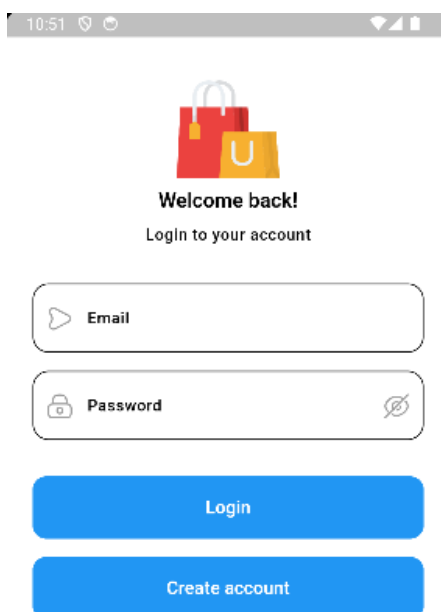
E-mail: [consumer@gmail.com](mailto:consumer@gmail.com)

Adgangskode: [Passcode1234?](#)

## Log ind eller Register bruger

For at logge ind på din konto, Indtast din e-mailadresse og adgangskode, derefter klik på Log in knappen.

For at registrere en ny konto, indtast dit fornavn, efternavn, telefonnummer, adresse, e-mailadresse og ny adgangskode, derefter klik på Create Account knappen.



10:51

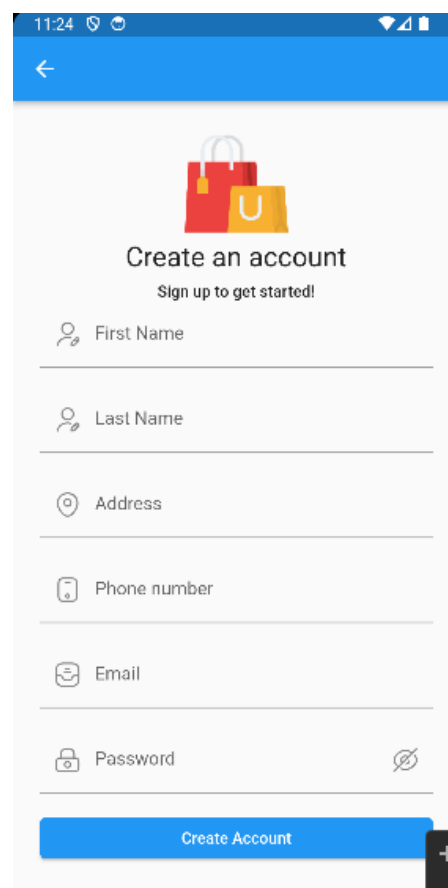
Welcome back!  
Login to your account

Email

Password

Login

Create account



11:24

Create an account  
Sign up to get started!

First Name

Last Name

Address

Phone number

Email

Password

Create Account

## Find produkt

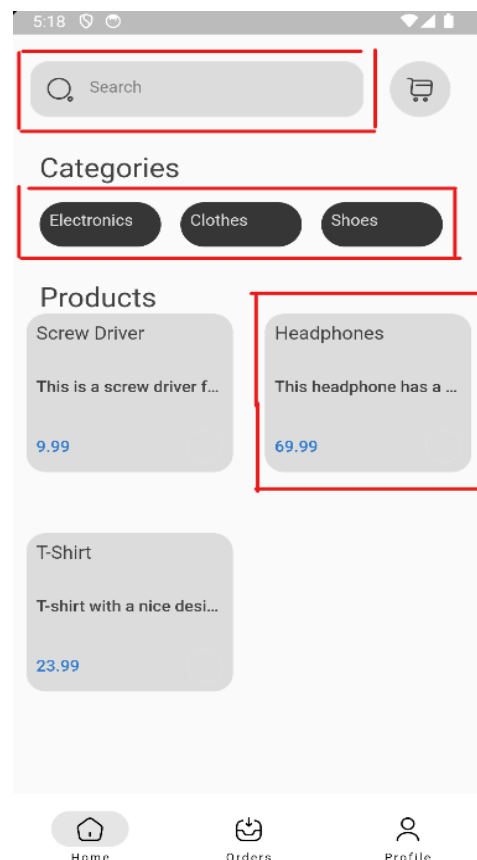
For at kunne se produkterne, skal man navigere til home screen ved at trykke på home knappen i BottomNavigationBar. Denne side indholder en liste over produkter og dets detaljer såsom navn, producent og pris. Øverst på siden kan du finde et søgefelt og kategori-tags til at filtrere produkterne ud fra dit behov.

## Se produkt detaljer

For at se produkt detaljer tryk på ønskede produkt card, så vil du blive navigeret til produktsiden. For at vende tilbage skal du trykke på Left Arrow icon knappen.

## Tilføj produkt til kurv

For at tilføje et produkt til kurven, skal man vælge et produkt ved at trykke på det, vil blive navigeret til produktdetaljer side, angive et produktmængde, og til sidst trykke på Add to Cart knappen.



5:18

Search

Categories

Electronics Clothes Shoes

Products

Screw Driver  
This is a screw driver f...  
9.99

Headphones  
This headphone has a ...  
69.99

T-Shirt  
T-shirt with a nice desi...  
23.99

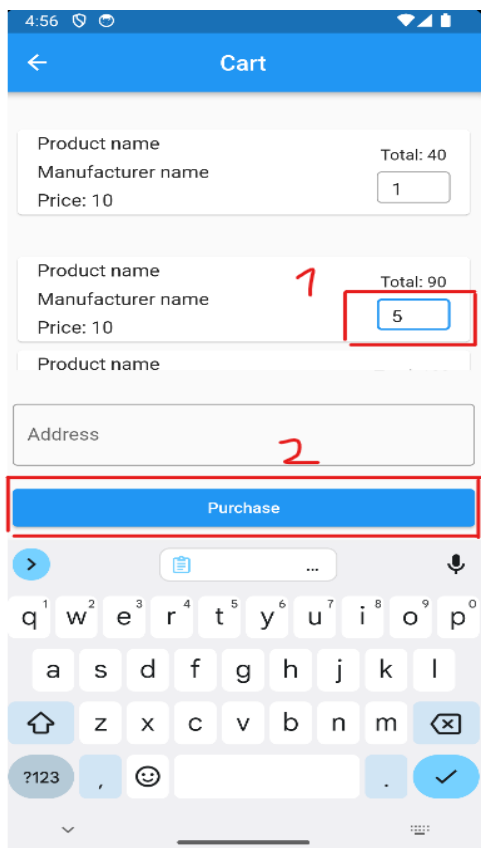
Home Orders Profile

## Kurv

For at se kurven skal man navigere til startside, og klikke på kurv-ikonet øverst til højre på siden. Denne handling fører dig til kurvside.

## Redigere vare i kurven

For at ændre varemængden skal man navigere til kurvsiden. Juster den aktuelle varemængde til din foretrukne værdi.

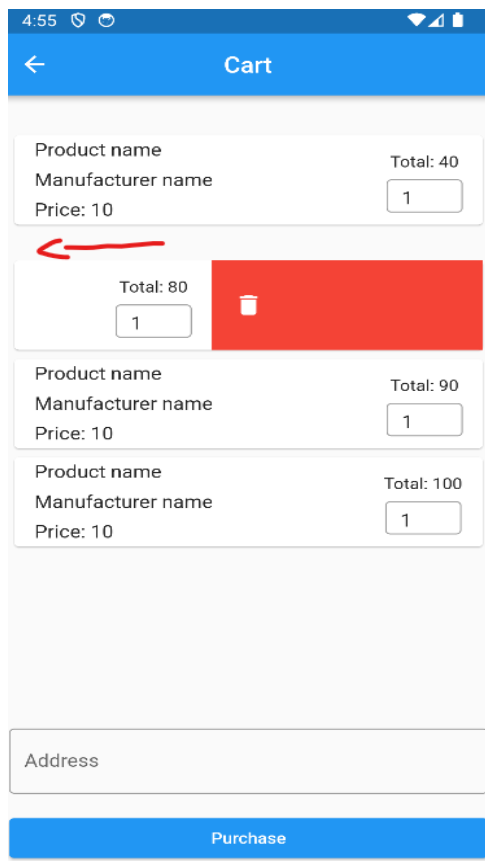
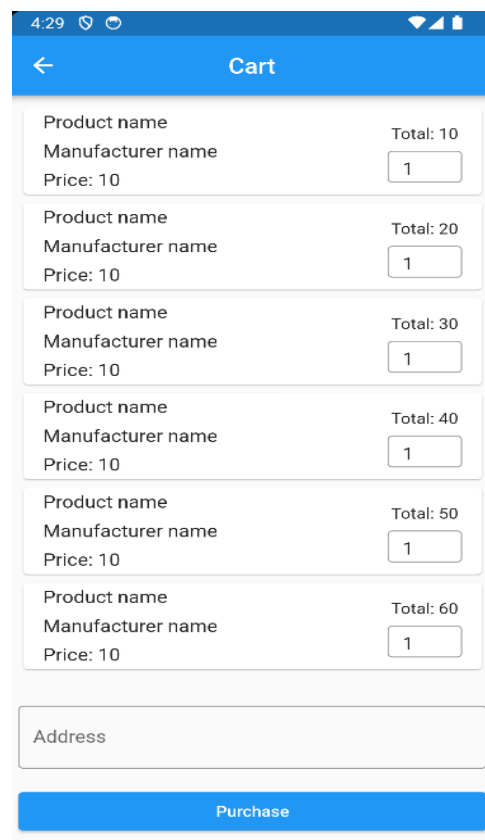


## Slet vare i kurven

For at slette en vare fra kurven, skal man stryge til venstre på varen.

## Afgive ordre

For at afgive en ordre, skal du navigere til kurvsiden. Skriv ned den ønskede destination for afsendelse af ordren til og tryk på købsknappen.

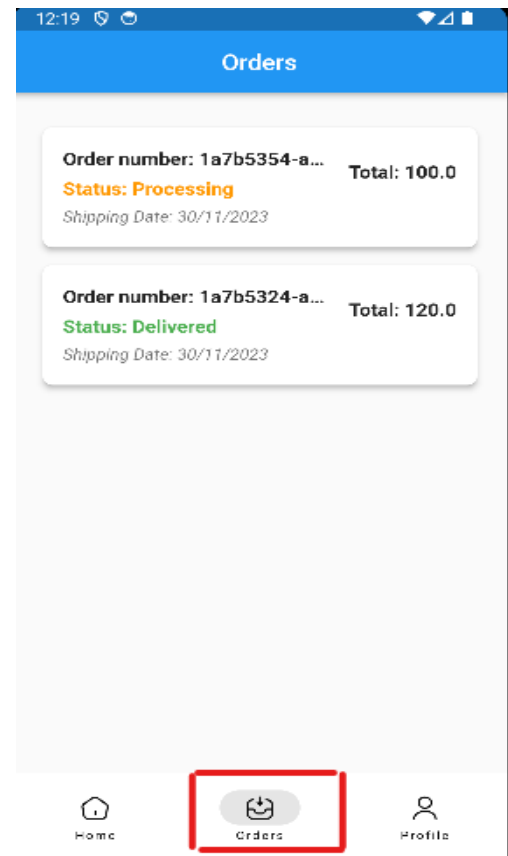
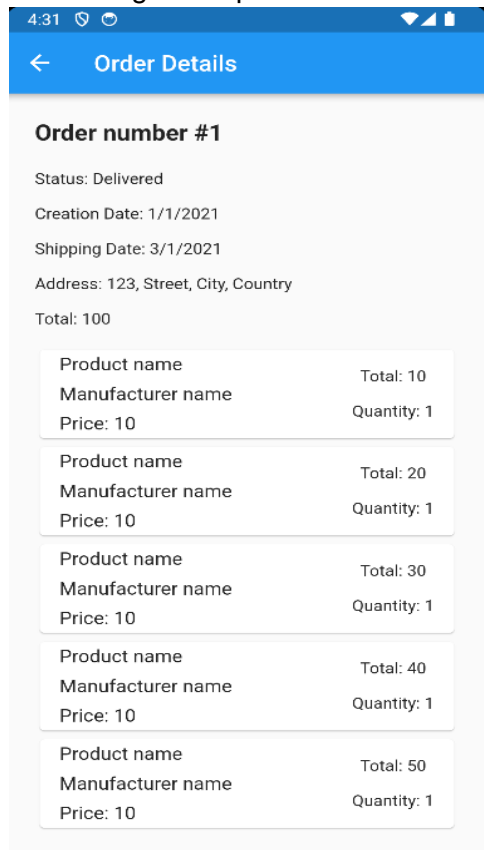


## Se ordrer

For at se ordrene skal man navigere til ordresiden.

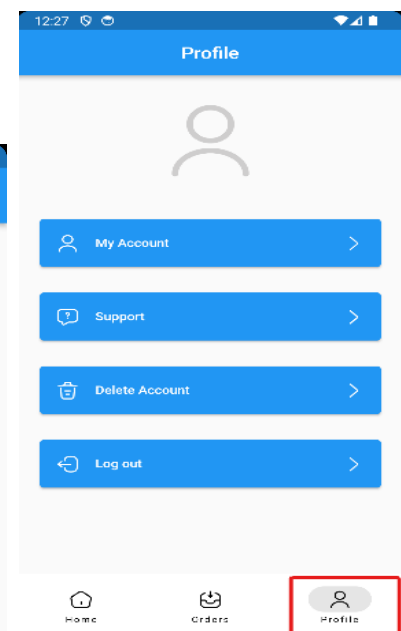
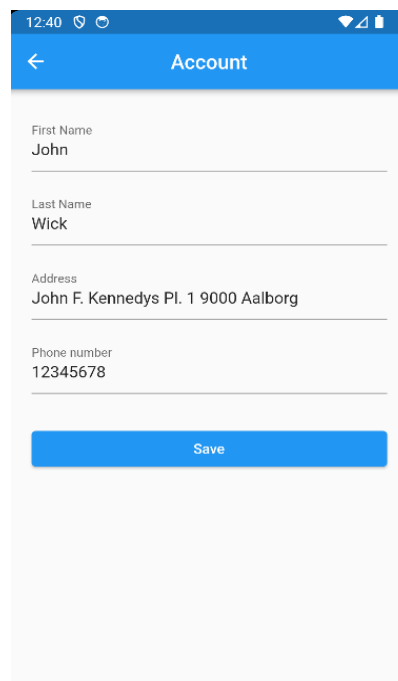
## Se ordre detaljer

For at se ordre detaljer tryk på ønskede ordre card, så vil du blive navigeret til produktsiden.



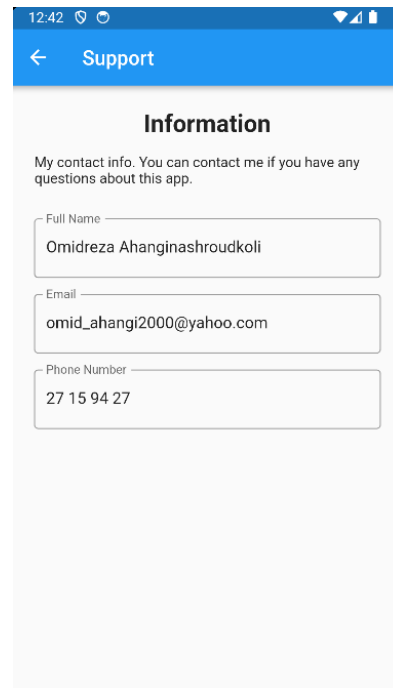
## Redigere konto

For at ændre kontoen skal man navigere til profilsiden. Klik på kontoknappen, juster de aktuelle oplysninger til din foretrukne.



## Support

Supportsiden indeholder mine kontaktoplysninger.



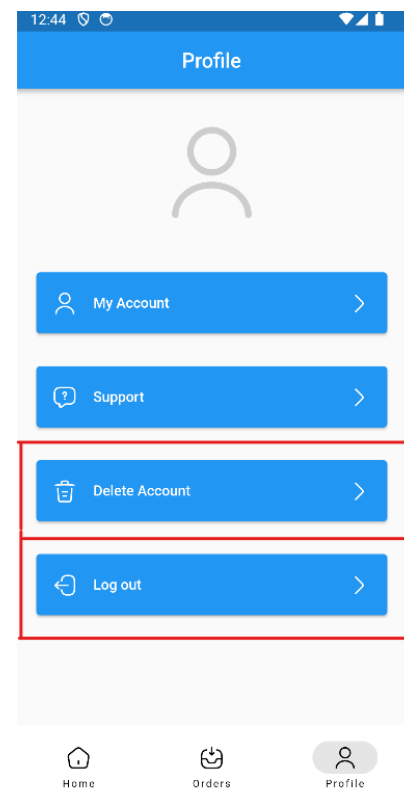
The screenshot shows the 'Support' page of the ShopXpress app. At the top, there is a blue header with a back arrow and the word 'Support'. Below this is a section titled 'Information' with a subtitle: 'My contact info. You can contact me if you have any questions about this app.' There are three input fields: 'Full Name' containing 'Omidreza Ahanginashroudkoli', 'Email' containing 'omid\_ahangi2000@yahoo.com', and 'Phone Number' containing '27 15 94 27'.

## Slet konto

For at slette kontoen skal du navigere til profilsiden og trykke på knappen Slet konto.

## Log ud

For at logge ud af kontoen skal du navigere til profilsiden og trykke på knappen Log ud.





# KONKLUSION

Det endelige produkt opfylder for det meste de oprindeligt fastsatte projektmål og de specificerede applikationskrav. Brugere kan med fordel bruge produktet til at udforske en bred vifte af produkter og uden besvær foretage køb, alt sammen fra deres hjem. Dette løser behovet for traditionelle fysiske butikker, herunder at navigere i trafik, navigere i omfattende produktudvalg og udholde lange kassekøer. Både forbrugere og administratorer kan trygt engagere sig i systemet, forsikret om, at deres data er sikkert opbevaret, og at systemet opretholder et højt niveau af tilgængelighed og stabilitet. Applikationens sikkerhed opfylder moderne standarder og implementerer robuste foranstaltninger såsom HTTPS, adgangskodekryptering, authentication og autorisation. Ydermere bidrager udnyttelsen af sikre hosting Services som Azure til den overordnede sikkerhed af systemet. Givet mere tid, som jeg nævnte i procesrapporten i afgrænsning afsnit, ville jeg have oprettet dedikerede frontend-administratorer. I øjeblikket skal systemets administratorer bruge Swagger til databaseændringer, selv om det er effektivt under udvikling, er det ikke egnet til produktionsformål. Givet muligheden for udvidet udvikling, vil implementering af en dedikeret administrativ frontend ikke kun forbedre brugeroplevelsen, men også tilpasse mere til bedste praksis for produktion.





# KIDELISTE

Web API : <https://shopxpressapi.azurewebsites.net/index.html>

Azure DevOps:

[https://dev.azure.com/omid0152/ShopXpress/\\_boards/board/t/ShopXpress%20Team/Epics](https://dev.azure.com/omid0152/ShopXpress/_boards/board/t/ShopXpress%20Team/Epics)

Versionsstyling

1. Rescuer: [https://dev.azure.com/omid0152/ShopXpress/\\_git/Resources](https://dev.azure.com/omid0152/ShopXpress/_git/Resources)
2. API : [https://dev.azure.com/omid0152/ShopXpress/\\_git/ShopXpress.API](https://dev.azure.com/omid0152/ShopXpress/_git/ShopXpress.API)
3. APP:  
[https://dev.azure.com/omid0152/ShopXpress/\\_git/ShopXpress\\_App?path=%2F&version=GBmaster&a=contents](https://dev.azure.com/omid0152/ShopXpress/_git/ShopXpress_App?path=%2F&version=GBmaster&a=contents)

Flutter APK-filen: [https://drive.google.com/drive/folders/1\\_I9-UfUAL44c8V0AEctdwByp0UTexDXz?usp=sharing](https://drive.google.com/drive/folders/1_I9-UfUAL44c8V0AEctdwByp0UTexDXz?usp=sharing)