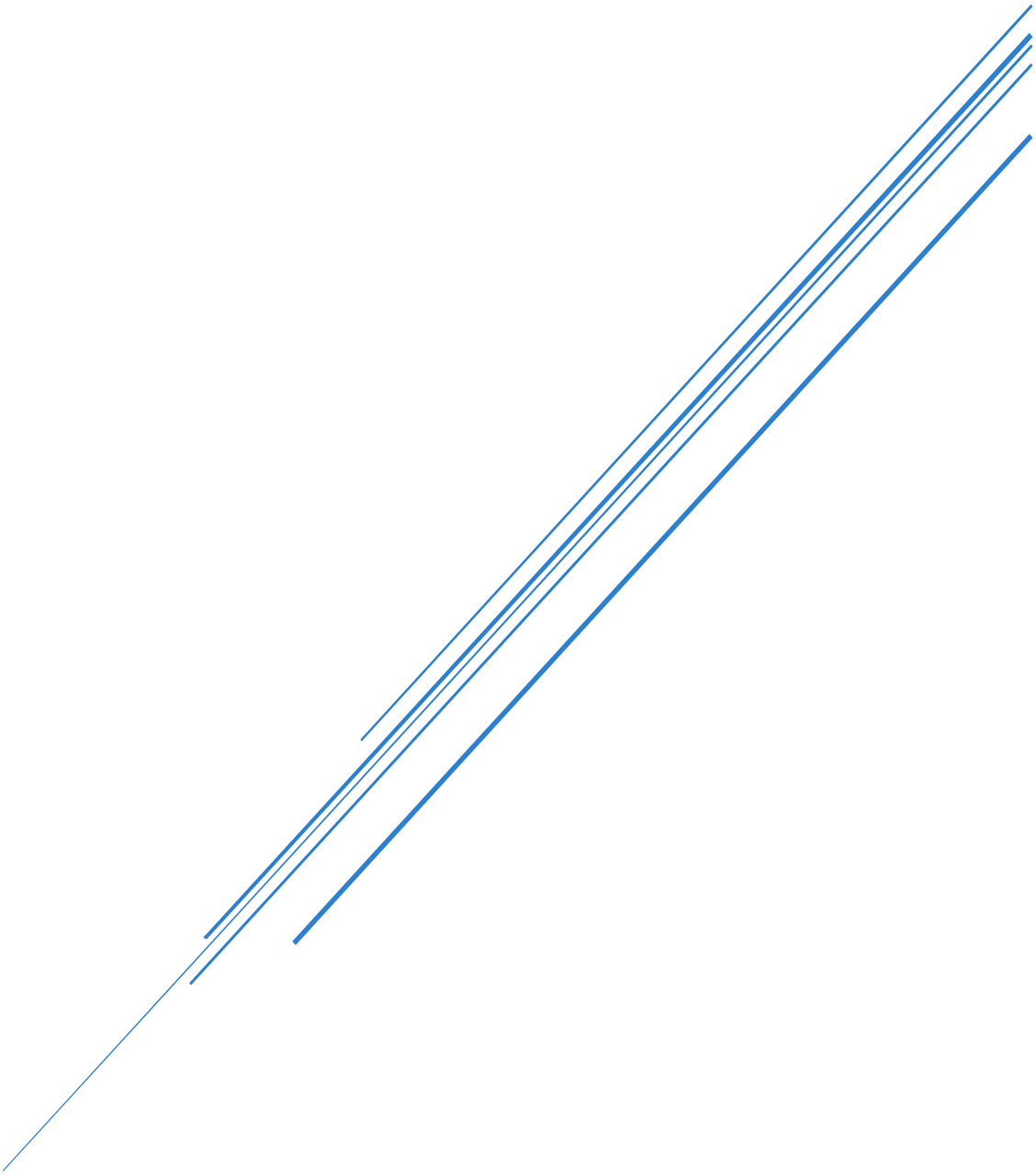
INLÄMNING 3

Software Design Description (SDD)



Maria Schillström

2024-04-02

Software Design Description (SDD)

1. Systemöversikt

Applikationen är en MVC-baserad webbapplikation som använder en trelagersarkitektur (Presentation, Applikation och Data) för att hantera information kopplat till prenumeranter. Syftet är att skapa en grundstruktur med tydlig fördelning, infrastruktur och CI/CD-stöd. Applikationen är uppsatt för drift i Azure-miljö och stöder framtida funktionalitet såsom bildlagring och användarhantering.

Funktionaliteten omfattar just nu:

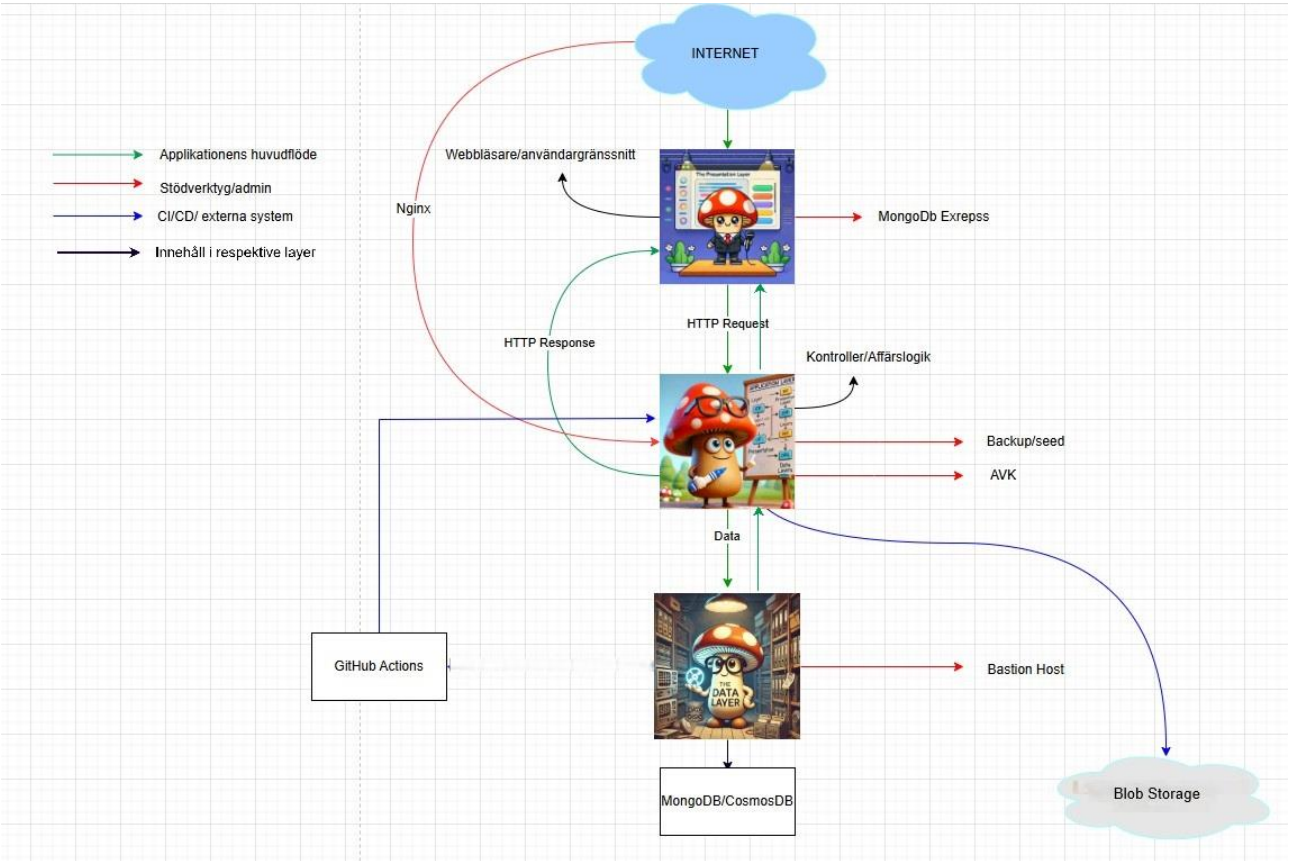
- Visa startsida och integritetspolicy
- Hantera "subscribers"

För framtida mål, vänligen se avsnitt 5. Förbättringar.

2. Arkitekturbeskrivning

▶ Visuell representation

Applikationen följer 3-skiktsarkitektur enligt modellen Presentation → Application → Data Layer.



▶ **Beskrivning av varje lager**

- **Presentation Layer**: Ansvarar för gränssnittet (t.ex. Razor Views), tar emot användarens input och presenterar data.
- **Application Layer**: Innehåller affärslogik, kontroller och services. Hanterar begäran från presentation och kommunicerar med datalagret.
- **Data Layer**: Kommunicerar med MongoDB eller CosmosDB via repository. Ansvarar för hämtning, lagring och uppdatering av data.

▶ **Stödkomponenter:**

- **Nginx**: Reverse proxy mellan Internet och applikationen.
- **Azure Bastion**: SSH-anslutning för administration.
- **MongoDB Express**: Webbaserat GUI för databasinsyn.
- **GitHub Actions**: CI/CD-pipeline för deployment.
- **Azure Key Vault (AVK)**: Säker lagring av hemligheter.
- **Blob Storage**: Används för lagring av publika bilder i applikationen.
- **CosmosDB** : Används för att lagra och hantera strukturerad data i applikationen

▶ **Teknikstack:**

Applikationen är byggd med .NET 9 och C# i en MVC-struktur med Razor Pages. Utvecklingen har genomförts i **Visual Studio Code** på en Windows-miljö. För deploy används **GitHub Actions**, och applikationen körs på en **virtuell Linux-baserad server i Azure**.

Databasen som används är **MongoDB**, och applikationen är även förberedd för **Azure CosmosDB** för framtida skalbarhet. För lagring av bilder används **Azure Blob Storage**.

Konfiguration hanteras via **appsettings.json**, och känsliga uppgifter, som **connection strings**, hanteras säkert via **Azure Key Vault**.

Applikationen är tillgänglig via en **Nginx reverse proxy**, och en **Bastion Host** används för säker åtkomst till VM:n. CI/CD-pipelinen med **GitHub Actions** möjliggör automatiska uppdateringar vid kodändringar, vilket säkerställer smidig och kontinuerlig integration.

3. Designmönster och implementation

▶ Använda designmönster:

- **Repository Pattern**: Separering av datalager och logik.
- **Service Pattern**: Tydlig uppdelning av affärslogik i Services.
- **Dependency Injection**: Beroenden skickas in via konstruktorer för att öka testbarhet och löskoppling.

Exempel; I koden nedan så vill jag använda något som uppfyller ett visst kontrakt- men jag bryr mig inte om exakt hur det är implementerat.

Det påminner lite om polymorfism via interface och abstrakta klasser med den skillnaden att;

- Med abstrakta klasser/interface styr du vad objektet kan göra.
- Med Dependency Injection styr du även hur det skapas och injiceras automatiskt av ramverket(till exempel .NET).
- I denna applikation används DI för att injicera tjänster som ISubscriberRepository och ISubscriberService i controllers och services.

▶ Dependency Injection i praktiken:

```
bool useMongoDb =
builder.Configuration.GetValue<bool>("FeatureFlags:UseMongoDb");

if (useMongoDb)
{
    builder.Services.AddSingleton<ISubscriberRepository,
MongoDbSubscriberRepository>();
}
else
{
    builder.Services.AddSingleton<ISubscriberRepository,
InMemorySubscriberRepository>();
}
builder.Services.AddScoped<INewsletterService, NewsletterService>();
```

▶ **Kodexempel:**

```
public class NewsletterService : INewsletterService
{
    private readonly ISubscriberRepository
_subscriberRepository;

    public NewsletterService(ISubscriberRepository
subscriberRepository)
    {
        _subscriberRepository = subscriberRepository;
```

Konfigurationsinställningar hanteras via appsettings.json och miljöspecifika varianter. Dessa laddas in via IConfiguration.

4. Konfiguration

▶ **Hantering av hemligheter:**

Känsliga uppgifter som till exempel connection strings hanteras via:

- Azure Key Vault (produktion)
- appsettings.Development.json eller miljövariabler (utveckling)
- **appsettings.json** innehåller konfigurationen för **MongoDB**, som lagrar **connection strings** och andra nödvändiga inställningar
- **Azure Key Vault** används för att lagra och säkert hämta känsliga uppgifter som **API-nycklar** och **hemliga tokens**. Applikationen hämtar dessa värden genom en säker URL som refererar till **Key Vault**:

```
"MongoDb": {
    "ConnectionString":
"mongodb://{username}:{password}@{hostname}:{port}",
    "DatabaseName": "cloudsoft",
    "SubscribersCollectionName": "subscribers"
  },

"AzureKeyVault":{
 "KeyVaultUri": "https://{keyvaultname}.vault.azure.net/"
```

5. Förbättringar och framtida utveckling

- **Lägga till användarhantering**
    - Administrativ roll
    - Autentisering
- **Bygga ut fler funktioner**
    - Söka efter svampar
    - Recept
    - Bokningsmöjlighet för t.ex svampkonsult
    - Boka svampträffar
    - Ladda upp egna bilder av svampar
    - Karta för att registrera ovanliga fynd med mera
    - CRUD
- **Centralisera loggning och felhantering**
    - Implementera ett inloggningsbibliotek för att samla alla loggar.
    - Hantera fel med try-catch block, och säkerställ att alla fel loggas på ett strukturerat sätt.

Begränsningar:

- Applikationen är för närvarande endast tillgänglig **lokalt** och har **inte publicerats ännu**.
- **NGINX** och **Azure Bastion** fungerar inte som förväntat i den nuvarande uppsättningen och behöver ytterligare konfiguration för att fungera korrekt.
- Vissa funktioner, som **användarhantering (autentisering och roller)**, är ännu inte implementerade.
- Applikationen saknar **fullständig frontend** och det finns inga enhetstester implementerade.