aaaService pattern, SOLID, single, open/closed, l principle , i principle, dependency inversion

**Azure funktioner att lära sig**

✅ **Azure Web App** Kör din webbapplikation (t.ex. ASP.NET Core, Node.js, m.m.).

🗃 **Azure SQL Database** Sparar data som användare, projekt, länkar till bilder m.m.  
 *Tips: Välj “Free” SKU för test/mindre appar.*

🖼 **Azure Blob Storage** Sparar filer som bilder, dokument, PDF:er, profilbilder, m.m.

🔐 **Azure Key Vault** Molnbaserad och säker ersättning för appsettings.json – lagrar hemligheter, API-nycklar och connection strings för din app i **molnet**, utanför koden.

Alla kopplas via min egna kod och connectionstrings.

**Olika Systemarkitekturer.**

* Monolitisk system - Domani/Shared/Data osv massa lager. Så vi byggt i asp.net kursen.
* Service Oriented Architecture **SOA** - Splittrar upp det i CLient service som innehhåller testning och deras lager, Authentication och deras lager, procejt service och deersa lager osv. Separatte sulotions, så kör vi det på APIer, då är det alltså ett stort projekt enbart för ett ansvars område ochså kommunicerar du från huvud appen til lde olika apparna via api:er. (Används i stora enterprise system) Shared nuget packages?, **Tips**
* Cors - \* tillåt alla - tänk på om du får cors problem.

**Frågor til aktiketksuren**

* I Soa Shared nuget apckages?
* I Soa Git submodal? Arva in repos?
* Enterprise service bus? Är det så SOa arkitetkuren?
* Api gateway ? frontend ui service, react t.ex eller mvc med react, som då ha controllerns,
* Så tanken är att UI service panelerna jobbar med api gate wat och de andra service sulotionsen jobbar med Enterprise servicebus?
* Microservices,
* Mer om enterprise service bus, kommunicationen mellan SOAservices

**Google Remote Procedure Call (gRPC)**

är ett högpresterande, open source-kommunikationsprotokoll utvecklat av Google för typad, effektiv och snabb kommunikation mellan tjänster, ofta som ett alternativ till REST och JSON. gRPC använder HTTP/2 och Protobuf (Protocol Buffers) för att skicka och ta emot kompakta, strongly typed meddelanden. Istället för att manuellt bygga JSON-API:er definierar man ett kontrakt i en .proto-fil, som innehåller meddelanden (requests/responses) och tjänstemetoder. Filen kompileras till C#-kod som genererar både klient- och serverklasser. Servern implementerar metoder genom att overridea bas-klasserna som genereras från .proto-filen, medan klienten anropar dessa metoder via GrpcChannel, utan att bry sig om HTTP, routes eller JSON. gRPC passar utmärkt för kommunikation **mellan olika lösningar och system**, särskilt i mikroservices eller SOA-arkitektur, där varje modul har ett eget gRPC-API. Det fungerar som ett alternativ till REST-API mellan client/server och även som ett **komplement till en Enterprise Service Bus**, där gRPC sköter direkta synkrona anrop (t.ex. hämta användare) och ESB sköter asynkrona eventflöden (t.ex. användare skapad). Vanliga användningsområden är system-till-system-kommunikation, interna API:er mellan microservices, mobilappar, och maskiner som behöver snabb och säker kommunikation. gRPC fungerar fullt ut i Azure-miljöer, har stöd för TLS, och kan integreras med andra tjänster som Key Vault, API Management och Azure Service Bus. Det används mellan lösningar på ett typat och versionssäkert sätt, vilket gör det till ett kraftfullt verktyg för moderna distribuerade system.

**WCF - för legacy SOA system.**

Soap? Vad är de thur används det? , och vad är då basicHttpBinding och när används det? Wshttpbionding bättre för transaktuikner? SOAP över TCP bara internt?, netTcpBinding? netNamedPipeBinding intern i systemet?, rest och webHttpBinding? Custombinding? Man måste installera wcf dne är borttagen som manuel tänst, så wcf när man ska kommunicera med gamla system iaf.

URI - är schemat för URL:n



Vi ska då i html formuläret använda de specifika RESTful metoderna och inte filtrera hela bodyn när den kommit med en post request utan att faktist använda rätt sak till rätt ändamål.

**CRUD + HTTP-metoder (REST):**

* **Create:** POST /resource – Skapa nyty
* **Read:** GET /resource – Hämta data
* **Update:** PUT /resource/id – Uppdatera hela objektet
* **Update (delvis):** PATCH /resource/id – Uppdatera fält
* **Delete:** DELETE /resource/id – Ta bort objekt

Vi kan använda olika typer av query parametrar. [t.ex](http://t.ex) ID, kategorier, osv. Väldigt viltigt vi försöker använda oss av edetta och subkategorier osv. Vi kan även skicka med Cookies i Requesten.

## **APIER - CORS**

UserCors, AllowAnyOrgin, Anyheader, Any Method, vilket vi då kan begränsa av vilen urlk eller header eller metod som kan köras. Även göra en Policy, [t.ex](http://t.ex) du vill bara ha de tför ditt eget system eller/och bara en viss metod för get/post osv,

## **Använd rätt verktyg på rätt plats**

| **Lager** | **Teknik** |
| --- | --- |
| UI → Backend | ✅ REST |
| Backend → Backend (C#) | ✅ gRPC |
| Backend → Legacy system (SOAP) | ✅ WCF |
| Backend → Azure tjänster | ✅ REST |
| Realtid mellan tjänster | ✅ gRPC |
| Webbläsare → Realtid | ✅ SignalR |

### **🧠 Tänk på det så här:**

* **REST** = Universalspråk (webb, appar, integrationer)
* **gRPC** = Intern effektivitet (C# ↔ C#, microservices, streaming)
* **WCF** = Endast om du måste (gamla SOAP-system)
* **SignalR** = Webbläsare behöver realtid
* **Azure-tjänster** = REST är norm

**Azure for students**

Hitta azure for student, välj start with 100 dollar

**Azure Resource group**:

* Web Apps / App Services
* Static Web Apps
* Key Vaults
* Azure SQL Databases
* Blob Storage
* Functions
* API Management
* Application Insights
* och mycket mer...

### **🧱 Microservice-arkitektur i Azure – fokusområden**

| **Kategori** | **Tjänster att fokusera på** | **Kommentar** |
| --- | --- | --- |
| 🖥️ Frontend | ✅ **Static Web App** | Hostar din React/Vue-frontend, pratar REST med backend |
| 🔧 Backend-API | **✅ App Service (Linux) eller ✅ Azure Functions** | Varje microservice (auth, booking, event etc.) körs här – REST och/eller gRPC |
| 💾 Databaser | **✅ SQL Database, Cosmos DB** | En databas per microservice, eller delad om det passar |
| 🔐 Secrets | ✅ **Key Vault** | API-nycklar, connection strings lagras säkert här |
| 📩 Kommunikation | **✅ Azure Service Bus (event-driven), ✅ gRPC (intern service–till–service)** | REST till frontend, gRPC + events internt |
| 📊 Observability | **✅ Application Insights** | Loggar och övervakar varje microservice individuellt |
| 📂 Resurshantering | **✅ Resource Group** | Samlar alla resurser till ett projekt för enkel hantering |
| ☁️ Storage | **✅ Azure Storage Account (inkl. Blob Storage)** | För lagring av filer, bilder, dokument etc. som används av microservices |

### **🚀 GitHub + Azure Static Web Apps – Sammanfattning**

När du kopplar din **GitHub-repo** till **Azure Static Web Apps** så skapas ett automatiskt flöde:

1. ✅ **Du pushar kod till GitHub** (t.ex. main-branchen).
2. ⚙️ **GitHub Actions** körs automatiskt – bygger projektet med t.ex. npm run build.
3. 🌐 **Azure publicerar automatiskt** den nya versionen av din webbplats.
4. 🔄 **Varje gång du uppdaterar repot**, så uppdateras sidan live – ingen manuell deploy behövs.

Det är alltså **CI/CD (Continuous Integration/Deployment)** direkt inbyggt. Enkelt, smidigt och gratis för små projekt.

### **💌 Azure Email Communication Services**

* En **del av Azure Communication Services (ACS)**, lanserad som preview 2023.
* Låter dig **skicka e-post via ett API** – perfekt för t.ex. kontobekräftelser, notiser, nyhetsbrev.
* **Stödjer SMTP och REST API**.
* Kräver att du **verifierar domäner** och ställer in **DKIM, SPF, DMARC**.
* Skalar lätt, bra leveranssäkerhet och pålitlig infrastruktur via Microsofts nätverk.

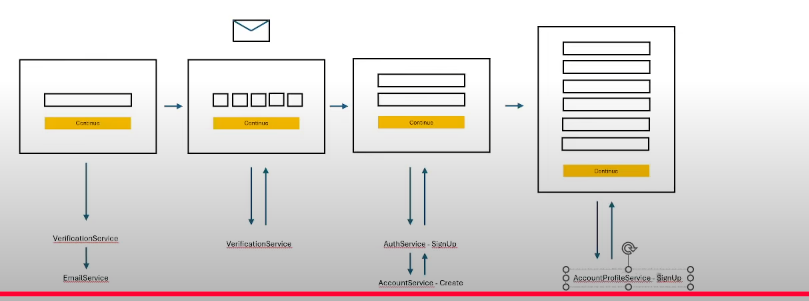
### **📞 Azure Communication Services (ACS)**

* En **plattform för realtidskommunikation** (inspirerad av Teams).
* Stödjer:  
  + **Röst- och videosamtal**
  + **Chatt**
  + **SMS**
  + **E-post (via Email Communication Services)**
* Du kan bygga appar som har inbyggd **kundsupport, möten, notifikationer m.m.**
* Används både i web, mobil och desktop – med SDK:er för flera plattformar.

### **🔄 Vad är Azure Service Bus?**

Azure Service Bus är en meddelandetjänst i molnet som används för att skicka data mellan system utan att de behöver prata direkt med varandra. Tjänsten stödjer både **queues** (en mottagare) och **topics med subscriptions** (flera mottagare), vilket gör den perfekt för lösningar där en händelse ska trigga flera olika tjänster. Det ger låg koppling, bättre skalbarhet och mer robusta system.

Bilagor



Flöde för email service och registrering med verifikation.

**Future mvp ideas**Connect their social media så it publishes on their different platforms they use then they create an event, maybe even create event on like facebook if possible which i think.

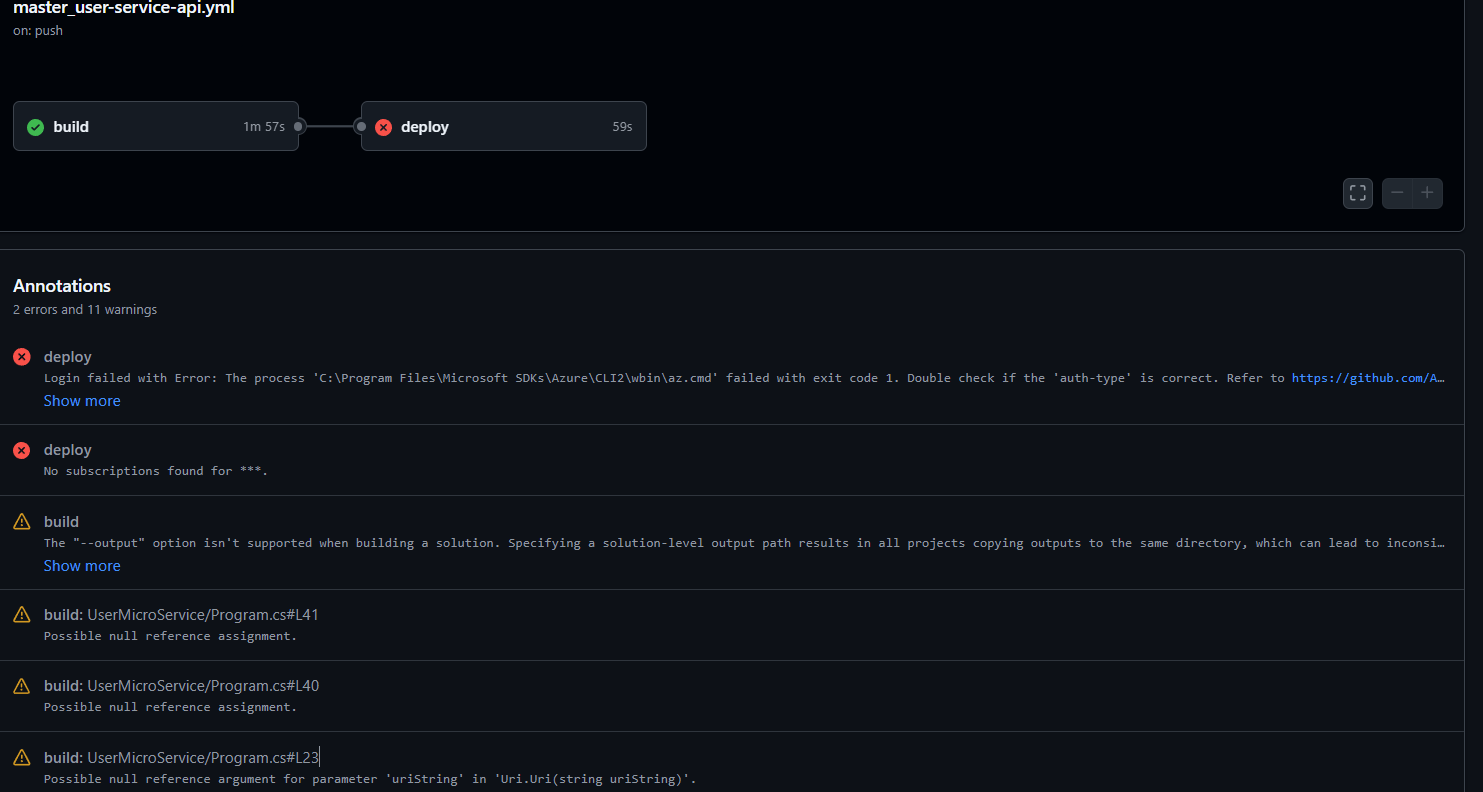
**Funktioner som måste finnas**

* Email verification + verifieras innan konto skapas så slipper vi confirm password.
* Forgot password?
* En ordentlig privacy
* En ordentlig Terms of condition
* En ordentlig contact som ska skicka meddelade till utvald email adress av ventrix.
* Mina Sidor med profil information **(Återkom med vad som måste finns där)**
* My events, liusta alla events man själv skapat oc hen skapa ny button.
* Dark mode ska funka
* GDPR och cookies fönster och settings ska funka, obligatorisk svarande första gången på login sen kan man ändra i settings inne i appen
* Notification system för updates som ske rpå follows, eller nya kommentarer [t.ex](http://t.ex) på de man följer eller de event man är attending to.
* Pages Timeline, Events, Attending, Following.
* Social media länk **eller inte?**
* Function apps och azure identity ska användas. Extern inloggning kommer vi vilja ha så gör det. Och Azure key vault.

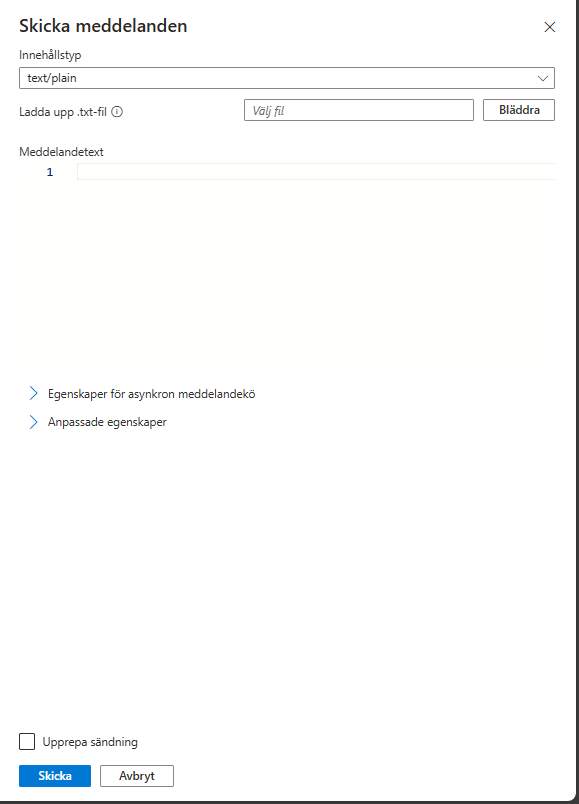
**Eventuella funktioner**

* Om du har löst allt med tidslinje, notiser, folowing, event skapade osv osv, så kanske ge dig på 2FA om det går.
* Kanske så man kan få notiser i appen + email då.

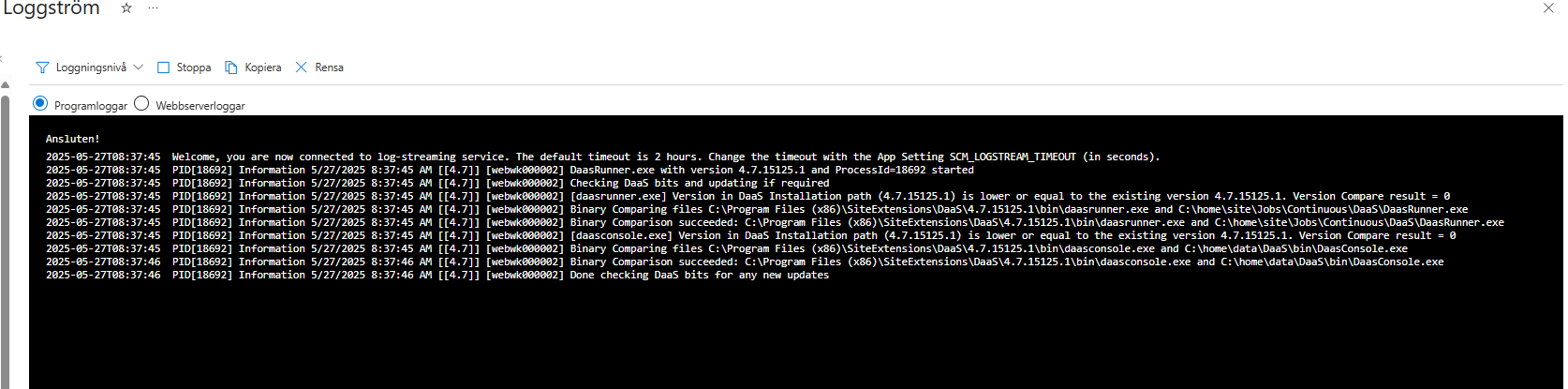
Andra bilagor



Github deplyment till azure



Testa skicka til lemail queue

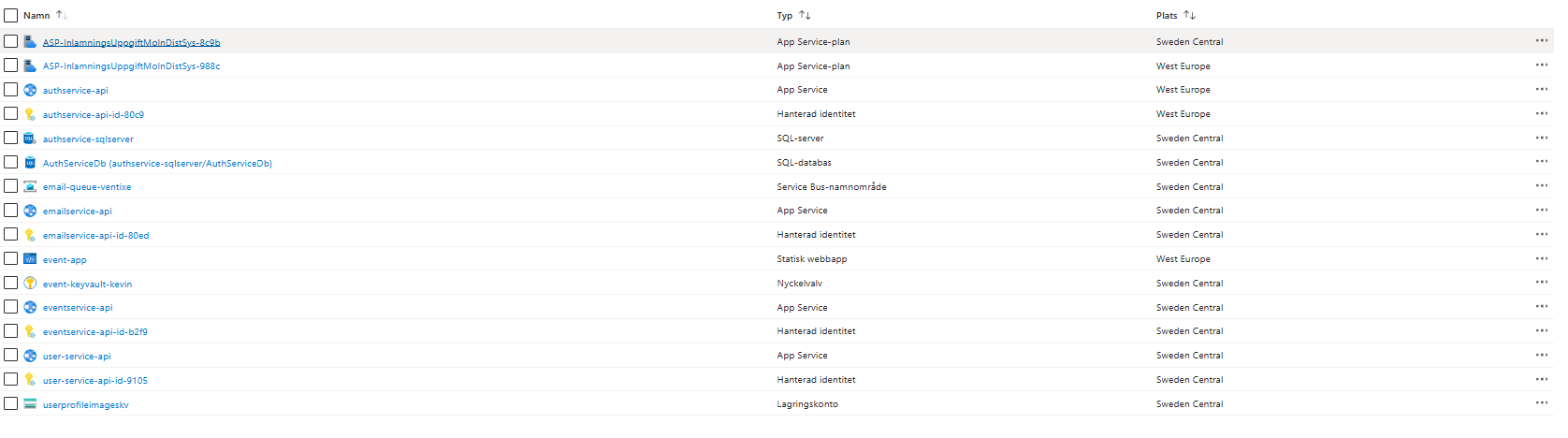


Loggströmmarna är väldigt effektiva och bra.

[JSON Web Tokens -](https://jwt.io/) [jwt.io](http://jwt.io)

Token decoder

Detaljer att fixa för en komplett MVP. - En av de slutgiltiga testerna.



Resurserna i resursgruppen.