Microchat

La chat a microservizi

Febbraio 2022 - Luca Deluigi, Simone Magnani, Thomas Angelini

Obiettivi del progetto e casi d’uso

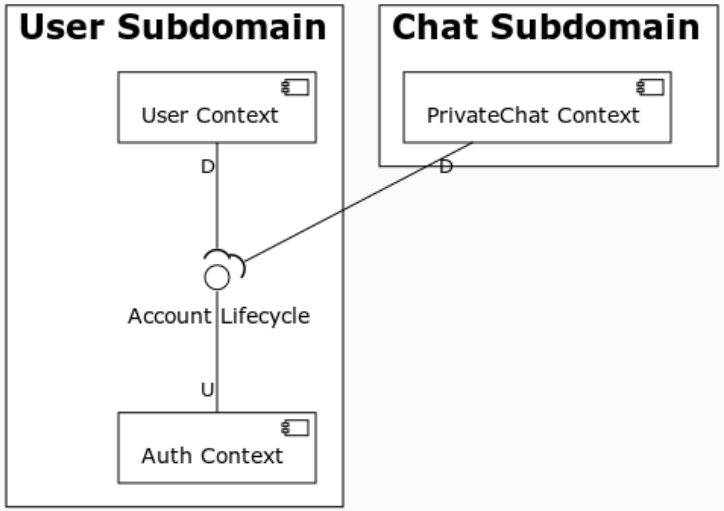
- Creazione di un servizio per l’instant messaging
  - La ricerca di un utente per poter iniziare il dialogo
  - Possibilità di dialogare con un utente attraverso messaggi testuali
- Il sistema sarà a microservizi che comunicano ad eventi asincroni (scambio di messaggi)
- L'applicazione sarà scalabile orizzontalmente in proporzione al numero di utenti
- Il protocollo di comunicazione client-server sarà bidirezionale e real time
- Integrazione nel progetto di principi e tecnologie DevOps

Domain Driven Design

- Principi e linee guida per affrontare domini complessi, applicando una metodologia Agile possibilmente con più team di sviluppo
- Step (per ogni iterazione di sviluppo):
  a. *Knowledge Crunching*: analisi dei Requisiti con gli esperti del dominio
  b. *Ubiquitous Language*: termini e significato all'interno del dominio
  c. *Context Map*: sottodomini, contesti (*bounded context*) e le loro relazioni
  d. *Modelli del dominio* (sincronizzati con il codice)
- Pattern tattici importanti per i microservizi
  a. *Aggregati e Aggregate Root*
  b. *Repository*
  c. *Eventi del dominio*

Context Mapping

- **User Subdomain (*supporting*)**: gestione dell'utente divisa in:
  - *Auth Context*: gestione dell'account utente per autenticazione
  - *User Context*: gestione del profilo utente
- **Chat Subdomain (*core*):** gestione delle interazioni via chat tra utenti
  - *PrivateChat Context:* messaggistica istantanea tra due utenti. Il tempo è gestito con UTC.

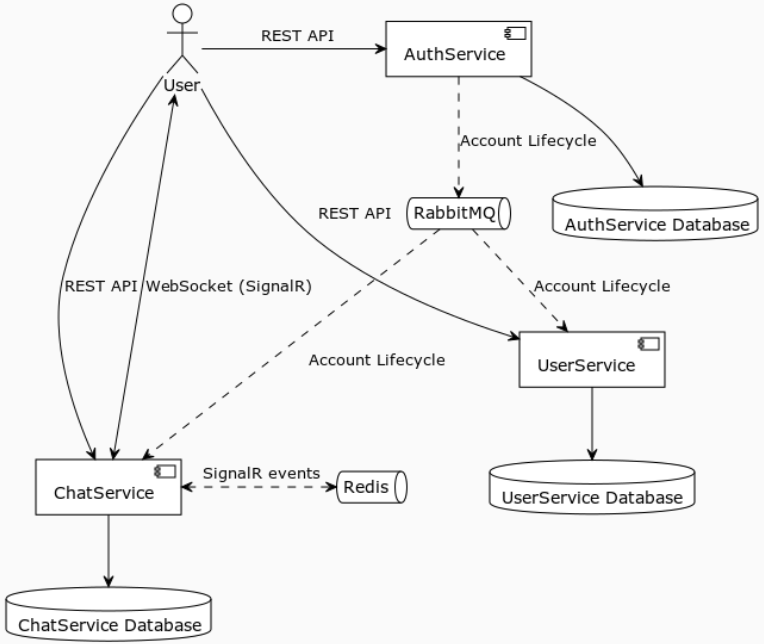


Design Architetturale: Microservizi

- Mapping 1 a 1 dai contesti ai microservizi
- Classificabile come Enterprise Application Integration System
  - Sottocategoria dei Distributed Information Systems
- Esempio di applicazione multi-scala del Single Responsibility Principle

Componenti Architetturali

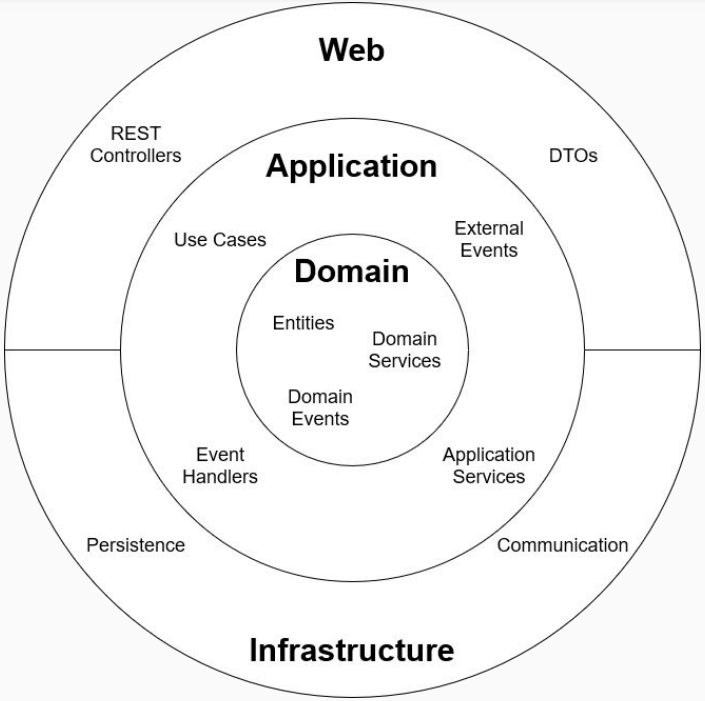
- Microsoft SQL Server: Database (uno per ogni servizio)
- RabbitMQ: Event Bus
- SignalR: Comunicazione bidirezionale client-server
- Redis: Backplane pub/sub per SignalR



Design Architetturale: Clean Architecture

Design di ogni singolo microservizio. Ogni strato dipende dagli strati più interni

- **Dominio** contiene le entità e le logiche di dominio del servizio
- **Applicazione** espone i casi d'uso e gli handler per gli strati più esterni, inoltre interagisce con il Dominio
- **Infrastruttura** gestisce la persistenza e la comunicazione con i servizi esterni
- **Web** è il punto di accesso per i client (endpoint REST)



Tecnologie utilizzate per lo sviluppo backend

- C#
- EasyDesk.CleanArchitecture
  - ASP.NET Core per il layer Web e l'IoC
  - Rebus per le comunicazioni tra servizi
  - MediatR per CQRS e gli handler (comandi, query, eventi del dominio)
  - EF Core per comunicare con SQL Server
- AspNetCore.SignalR + AspNetCore.SignalR.StackExchangeRedis

Scalabilità (dei servizi)

- Comunicazioni tra servizi basate su scambio di messaggi asincrono
  - Eventi (publish/subscribe)
  - Comandi (command processor)
- Statelessness dei servizi by design
  - Fatta eccezione per il protocollo WebSocket/SignalR
- Replicazione
  - Delle istanze dei servizi (load balancing)
  - Dei dati tra contesti diversi (quindi tra microservizi diversi) tramite gli eventi, in lettura
- JSON Web Tokens

Replicabilità dei servizi

- RESTful design ed eventual consistency tra aggregati
  - Le richieste coinvolgono solitamente un singolo aggregato, mappati a risorse
  - Possibilità di supportare sharding
- CQRS
  - Proprietà ACID all'interno dell'aggregato per i comandi, usando le transazioni
  - Proprietà BASE per query ed eventi

Scalabilità (del software)

- DDD per l'*Administrative Scalability*
- Clean Architecture
- Inversion of Control + Reflection
- Middlewares
  - MediatR
  - ASP.NET Core
  - Rebus
  - SignalR

Dockerizzazione

Containerizzazione di ogni componente:

- Creazioni immagini riutilizzabili in diversi sistemi
- Supporto nativo per la creazione del Dockerfile tramite Visual Studio

Docker compose per la messa in opera del progetto:

- DNS per la comunicazione inter-componente
- Esposizione porte e mappatura all’esterno dell’ambiente virtuale

L’applicazione frontend

- Sviluppo in Angular 13 in visione object-oriented
- Divisione in componenti e servizi autonomi secondo MVVM
- Incapsulamento delle operazioni simili: un servizio Angular per ogni microservizio
- Separazione tra ambiente in produzione e sviluppo
- Conversione da UTC a tempo locale

Demo

Grazie per l’attenzione

Microchat - La chat a microservizi

Febbraio 2022 - Luca Deluigi, Simone Magnani, Thomas Angelini