

API

Die API basiert auf einer MVC + Repo Architektur. Dieses Architekturmuster unterscheidet zwischen:

- `[[#Models]]`, welche zwischen der API & App geteilt werden
- Views, welche lediglich Json der Models sind und daher nicht genauer beschrieben werden
- `[[#Controllers]]`
- `[[#Repos]]`

Controllers


Controllers stellen die Routes / Endpoints der API dar. Weiters wird hier die Auth middleware eingesetzt.

Repos

Repos sind der Knotenpunkt zwischen Interaktionen mit der Datenbank via EF Core und den Controllers.

MAUI App

Die App basiert auf einer `[[#MV(VS)VMS]]` Architektur. Dieses Architekturmuster unterscheidet zwischen:

- `[[#Models]]`, welche zwischen der API & App geteilt werden
- `[[#Views]]`
- `[[#(View States)]]`
- `[[#View Models]]`
- `[[#Services]]` Der Einsatz dieser Komponenten erfolgt mittels Dependency Injection welche durch die `IServiceCollection` der MAUI App bereitgestellt wird. 

Views

Pages

Die Pages der `[[#MAUI App]]` inkludieren:

- Chat
- Chat details

- Chats
- User Profile
- Login / Register
- Settings

Views

Views kommen in dem Rahmen des Projekts nur passiv zum Einsatz. Sie besitzen keine Logik oder eigenen View Models, noch werden sie von anderen angesprochen. Views dienen daher lediglich der Isolierung von wiederholt eingesetzten View Elementen. So wird z.B. der Chat List View unter der Chats Page eingesetzt um eine einheitliche und isolierte Darstellung eines Chats zu beschreiben.

(View States)

Ein View State beschreibt den Modus eines Views. Ein View (in diesem Kontext eine Page) kann somit ein Mal deklariert werden und trotzdem verschieden angezeigt werden.

Beispiel

Die Chat Details Page profitiert von diesem Schema am meisten und bietet sich als ein gutes Beispiel an. Sie kann in den Modi `PageMode.View`, `PageMode.Edit` oder `PageMode.New` geöffnet werden.

Im `PageMode.View` werden z.B. Felder als Labels angezeigt und können somit nur betrachtet aber nicht bearbeitet werden. Änderungen können somit nicht vorgenommen werden. Im `PageMode.Edit` werden z.B. Felder als Entries angezeigt und können somit bearbeitet werden. Änderungen können somit an existierenden Chats vorgenommen werden. Im `PageMode.New` werden z.B. Felder ebenfalls als Entries angezeigt und können somit bearbeitet werden. Ein neuer Chat kann somit vom User mit Informationen befüllt werden, bevor dieser erstellt wird.

View Models

View Models dienen lediglich der Logik welche in direkter Verbindung zum View steht. Ein View Model ändert so z.B. die zugehörigen `[[#(View States)]]` oder bietet visuelles Feedback für den User. Weiters bereitet es Daten auf, welche spezifisch für den zugehörigen View relevant sind.

Ein View Model teilt z.B. ein Model wie einen Chat so auf, dass die Messages auf eine lokale `ObservableCollection` zugewiesen werden, da der View somit die Addition oder Subtraktion von Messages mitbekommt.

Logik wie z.B. die Interaktion mit der `[[#API]]` wird auf `[[#Services]]` ausgelagert, da diese nicht in direkter Verbindung mit dem View steht.

Services

Services behandeln jegliche Logik welche der User nicht direkt visuell mitbekommt. Darunter die Transformation und das Error Handling bezüglich Interaktionen mit der API. Services werden durch ihre assoziierten Interfaces implementiert.

Ein Sonderfall ist hier der `HttpService`. Dieser ist ein Wrapper der standard Class `HttpClient`. Er kümmert sich um Serialisierung und beugt Socket exhaustion vor. Er wird von anderen Services neben dem `LocalDbService` eingesetzt.

Classes

Statics

Die Statics Class beinhaltet Felder welche zum start der App ausgefüllt sind, jedoch können diese im verlauf der runtime verändert werden. Diese Felder sind im NameSpace der App verfügbar und sollen das abrufen von global relevanten Informationen erleichtern. So wird zum Beispiel der `AppOwner`, welcher eine lokale Abwandlung des `AppUsers`, welcher den momentan angemeldeten `AppUser` representiert, gesetzt sobald dieser sich angemeldet hat.

```
public static class Statics
{
    #if ANDROID
        public static string RouteBaseHttp = "http://10.0.2.2:5225/api/";
        public static string RouteBaseHttps = "https://localhost:7116/api/";
    #elif WINDOWS
        public static string RouteBaseHttp = "http://localhost:5225/api/";
        public static string RouteBaseHttps = "https://localhost:7116/api/";
    #else
        public static string RouteBaseHttp = string.Empty;
        public static string RouteBaseHttps = string.Empty;
    #endif
    public static string LocalSQLiteConnection = "DataSource=mysharedddb;mode=memory;cache=shared";
    public static string DefaultEmail = "user1@mail.com";
    public static string DefaultPassword = "P455w0rd!";
    public static BearerToken? BearerToken { get; set; }
    public static AppOwner? AppOwner { get; set; }
}
```

Models

![[Pasted image 20240602211521.png]]

AppUser

Represents the standard User for both the [#API] and the [#MAUI App].

```
public class AppUser : IdentityUser<Guid>
{
    public ICollection<Chat> Chats { get; set; } = [];
    public ICollection<Message> Messages { get; set; } = [];
}
```

DTOs

Read

```
public class AppUser_Read
{
    public Guid? Id { get; set; }
    public string? DisplayName { get; set; }
    public string? UserName { get; set; }
}
```

Create

```
public class AppUser_Create
{
    public string Email { get; set; }
    public string Password { get; set; }
}
```

Maps

```
public class AppUser_AMProfile : Profile
{
    public AppUser_AMProfile()
    {
        CreateMap<AppUser, AppUser_Read>();
    }
}
```

Chat

Represents the standard Chat for both the [[#API]] and the [[#MAUI App]].

```
public class Chat
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
```

```
public ICollection<AppUser>? Users { get; set; } = [];  
public ICollection<Message_API>? Messages { get; set; } = [];  
}
```

DTOs

Create

```
public class Chat_Create  
{  
    public string Name { get; set; }  
    public string? Description { get; set; }  
    public List<Guid> UserIds { get; set; } = [];  
}
```

Read

```
public class Chat_Read  
{  
    public required Guid Id { get; set; }  
    public required string Name { get; set; }  
    public required string Description { get; set; }  
    public IEnumerable<AppUser>? Users { get; set; }  
    public IEnumerable<Message_API>? Messages { get; set; }  
}
```

Update

```
public class Chat_Update  
{  
    public Guid Id { get; set; }  
    public string Name { get; set; }  
    public string? Description { get; set; }  
    public List<Guid> UserIds { get; set; } = [];  
}
```

Maps

```
public class Chat_AMProfile : Profile  
{  
    public Chat_AMProfile()  
    {  
        // Rest -> Transition  
        // API  
        CreateMap<Chat, Chat_Read>();  
        // MAUI  
    }  
}
```

```

    CreateMap<Chat, Chat_Create>();

    // Transition -> Rest
    // API
    CreateMap<Chat_Create, Chat>();
    CreateMap<Chat_Update, Chat>();
    // MAUI
    CreateMap<Chat_Read, Chat>();
}
}

```

Message

Represents the standard Message for both the [[#API]] and the [[#MAUI App]].

```

public class Message
{
    public Guid ChatId { get; set; }
    public Chat Chat { get; set; }
    public Guid UserId { get; set; }
    public AppUser User { get; set; }
    public int MessageId { get; set; }
    public string Text { get; set; }
}

```

DTOs

Create

```

public class Message_Create
{
    public Guid ChatId { get; set; }
    public Guid UserId { get; set; }
    public string Text { get; set; }
}

```

Read

```

public class Message_Read
{
    public Guid ChatId { get; set; }
    public Guid UserId { get; set; }
    public AppUser User { get; set; }
    public int MessageId { get; set; }
    public string? Text { get; set; }
}

```

Maps

```
public class Message_AMProfile : Profile
{
    public Message_AMProfile()
    {
        CreateMap<Message, Message_Create>();
        CreateMap<Message, Message_Read>();

        CreateMap<Message_Create, Message>();
        CreateMap<Message_Read, Message>();
    }
}
```

Relations

User & Chats

```
builder.Entity<AppUser>(u =>
{
    u.HasKey(u => u.Id);

    u.HasMany(u => u.Chats)
        .WithMany(c => c.Users)
        .UsingEntity<Dictionary<Guid, object>>(
            "UserChat",
            j => j
                .HasOne<Chat>()
                .WithMany()
                .HasForeignKey("ChatId")
                .OnDelete(DeleteBehavior.Cascade),
            j => j
                .HasOne<AppUser>()
                .WithMany()
                .HasForeignKey("UserId")
                .OnDelete(DeleteBehavior.Cascade));
});
```

Message

```
builder.Entity<Message>(m =>
{
    m.HasKey(m => new { m.UserId, m.ChatId, m.MessageId });

    m.HasOne(m => m.User)
        .WithMany(u => u.Messages)
        .HasForeignKey(m => m.UserId)
        .OnDelete(DeleteBehavior.Cascade);
```

```
m.HasOne(m => m.Chat)
  .WithMany(c => c.Messages)
  .HasForeignKey(m => m.ChatId)
  .onDelete>DeleteBehavior.Cascade);
});
```

Abläufe

Chat Erstellen

App

API