Mjlsi

Software Documentation

Version 1.0

[**Project Overview**](#_heading=h.3dy6vkm) **3**

[Functional Modules](#_heading=h.1t3h5sf) 3

[Non-functional Modules](#_heading=h.4d34og8) 3

[**Built With**](#_heading=h.2s8eyo1) **3**

[Backend](#_heading=h.17dp8vu) 3

[Front End](#_heading=h.3rdcrjn) 3

[**System Architecture**](#_heading=h.lnxbz9) **3**

[Container Diagram](#_heading=h.1ksv4uv) 4

[Components Diagram](#_heading=h.z337ya) 5

[Database Diagrams](#_heading=h.2xcytpi) 7

Agenda 7

Committee 8

Jobs 9

Chat 10

Meetings 11

Organizations 12

Roles and rights 13

Tasks 14

Users 15

Configuration and Settings 16

Documents 17

Stakeholders 18

Votes 18

Directories 19

Notification 20

[**Development Setup**](#_heading=h.3o7alnk) **20**

[Prerequisites](#_heading=h.23ckvvd) 20

[Installation](#_heading=h.ihv636) 20

[**System Configurations**](#_heading=h.41mghml) **21**

[**Deployment**](#_heading=h.2grqrue) **22**

[Hardware Configuration](#_heading=h.vx1227) 22

[Software configuration](#_heading=h.3fwokq0) 22

# Project Overview

Mjlsi is a system to manage meetings, documents and tasks for medium and large enterprises and it includes the following aspects:

## Functional Modules

* Registration and Login
* Dashboard
* Meetings
* Decision
* Tasks
* Members
* Review rooms
* Conversations
* Settings and configuration
* Files
* Technical support

## Non-functional Modules

* Bi-language
* Security
* Email Service

# Built With

## Backend

* ASP.NET Core 6

## Front End

* AngularJS 13

# System Architecture

The system architecture consists of system components and the sub-systems developed that will work together to implement the overall system.

The system is built based on the concept of separating the back-end and frontend to enjoy the following benefits:

* Provides APIs for external integrations
* Allow multiple clients APIs requests for future expandability like developing mobile application for the system
* The ability to change the UI design without affecting the app engine
* Code organization and team management

The following diagram describes the system architecture starting from the main containers of the system, main components and database diagram.

## Container Diagram

The following is a container diagram that describes the cloud enabled architecture for the system. The system is divided into frontend and backend and uses the JWT (JSON Web Token) standard in securing the communication between the backend and frontend.

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

The system is also using ASP.NET Core Web APIs to create the cloud Endpoints.  
ASP.NET Core Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices. It is an ideal platform for building RESTful applications on the .NET Core Framework.

The system is also using ASP.NET Core Entity framework for standard communication between the different layers of the application and the database.

Entity Framework Core is a modern object-database mapper for . NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with many databases, including SQL Database (on-premises and Azure), SQLite, MySQL, PostgreSQL, and Azure Cosmos DB.

# 

## Components Diagram

This diagram is intended for internal developers and shows them how to organize and develop code. This diagram serves the following purposes:

* It describes the components or services of the system.
* It clarifies the relationships and dependencies between components.
* It provides a framework that shows how software development tasks can be distributed and delivered.

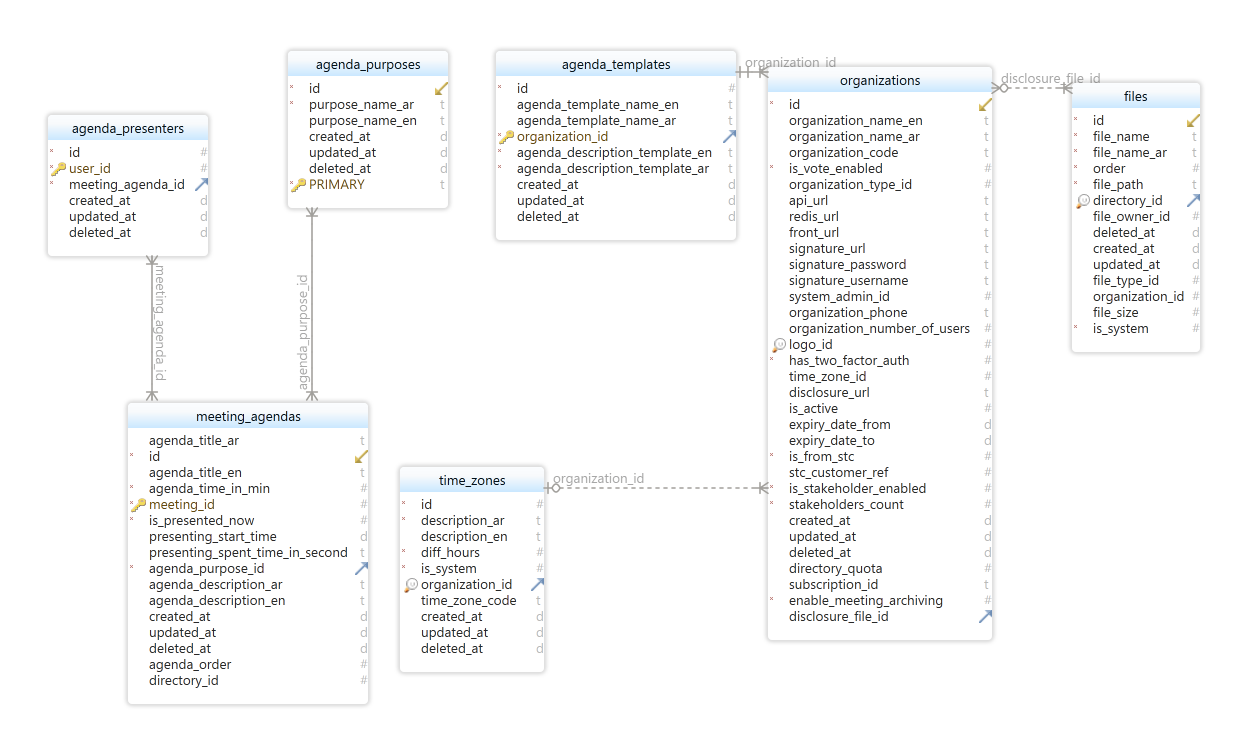
# 

The system is a data-driven application which needs to access the data residing in one or more other data sources. Most of the time data sources will be a database. these data-driven applications need to have a good and secure strategy for data access to perform the CRUD operations against the underlying database. One of the most important aspects of this strategy is the separation between the actual database, queries and other data access logic from the rest of the application. In our system, we need to separate the data access logic from the Controllers. The Repository Design Pattern is one of the most popular design patterns to achieve such separation between the actual database, queries and other data access logic from the rest of the application.

# 

## Database Diagrams

### Agenda



### Committee

### 

### 

### 

### 

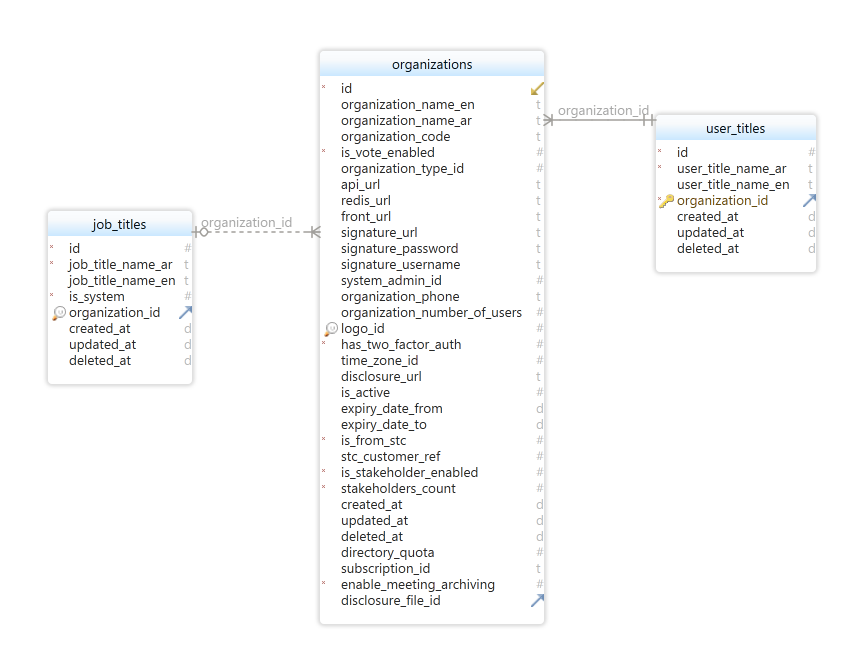
### 

### 

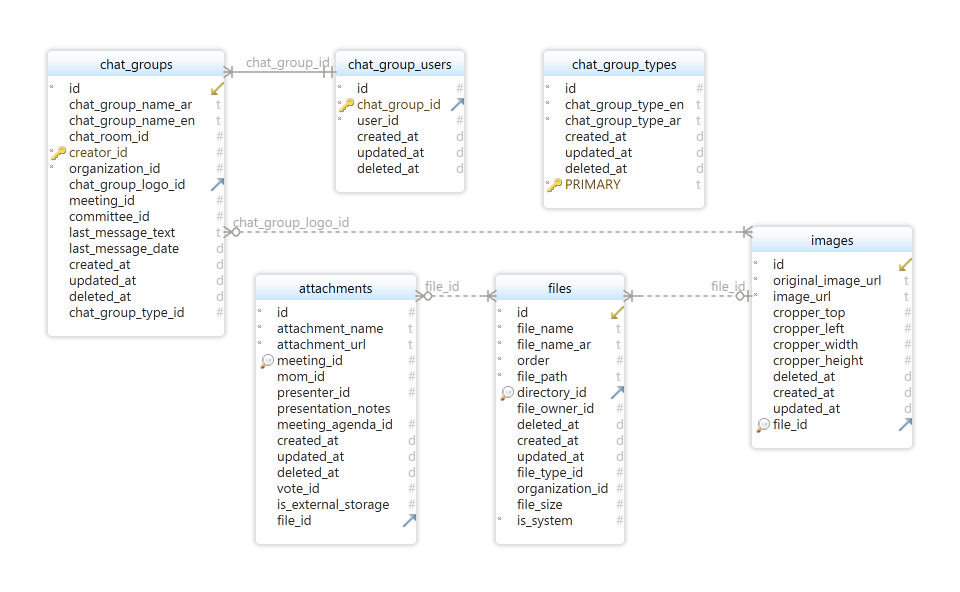
### 

### 

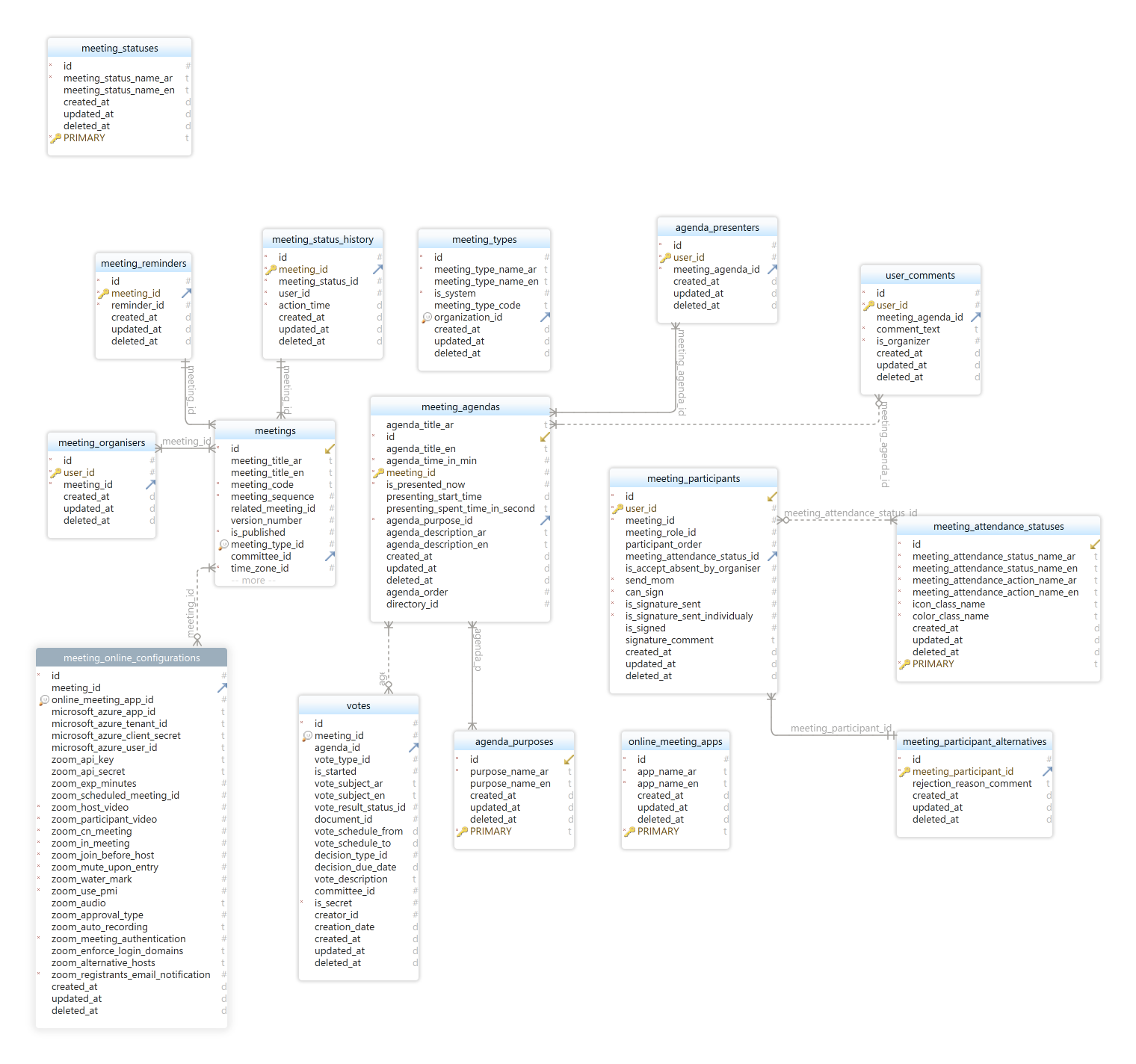
### Jobs



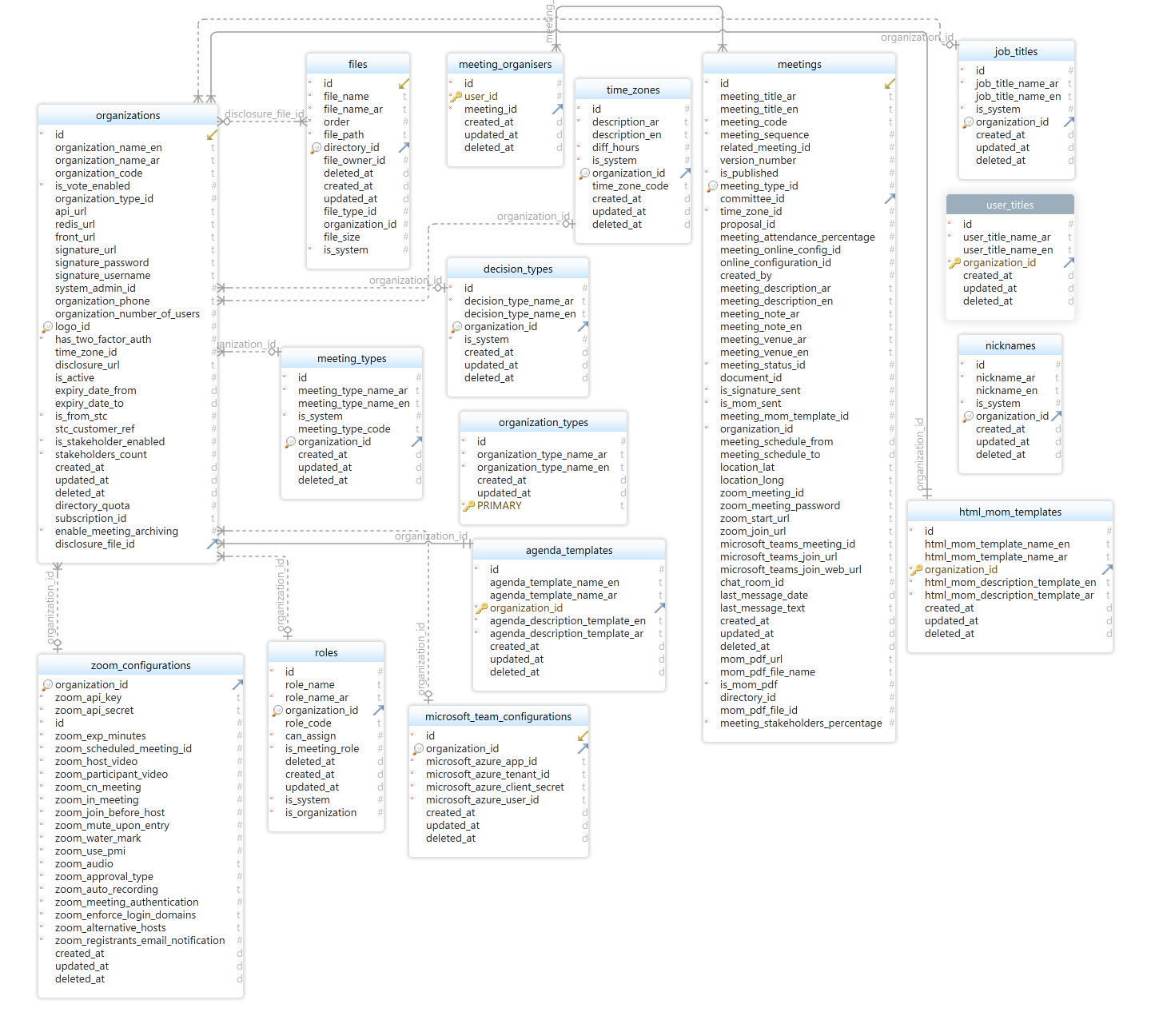
### Chat



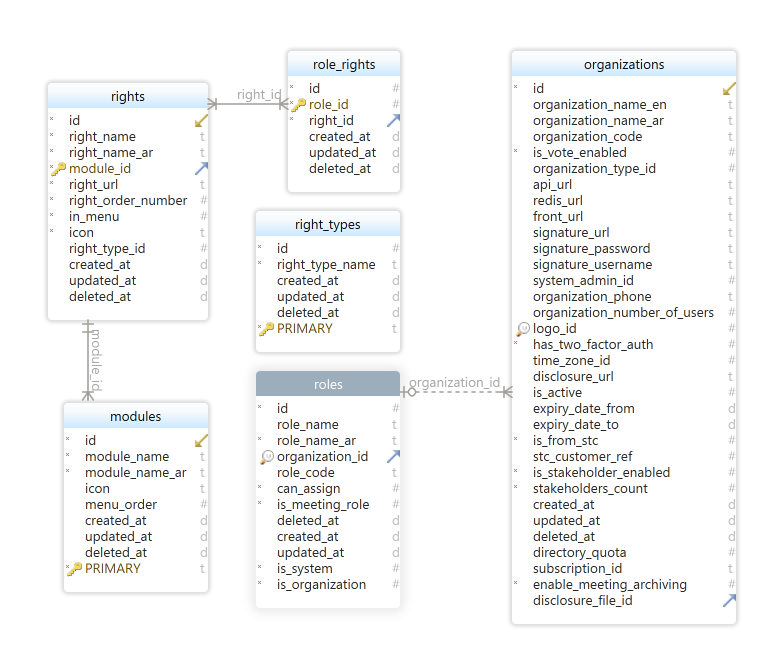
### Meetings



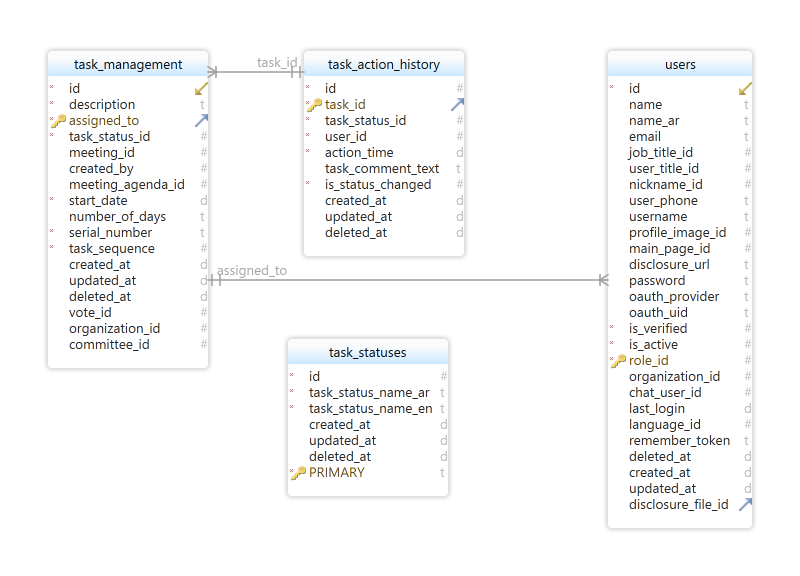
### Organizations



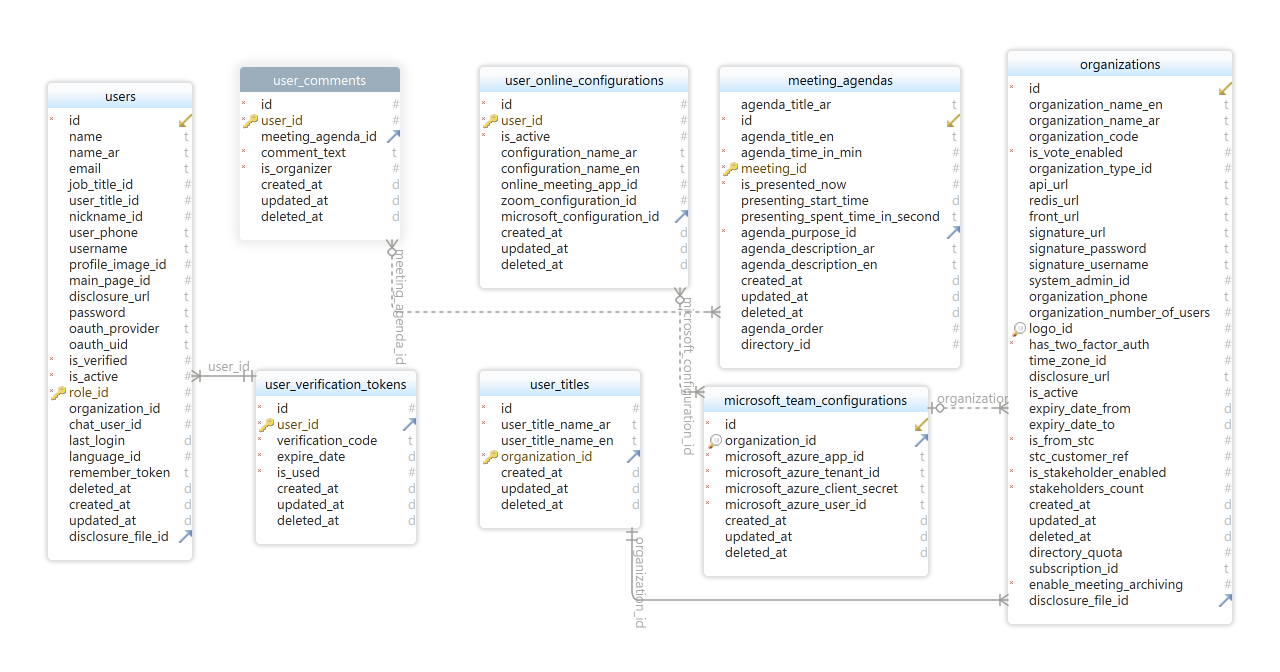
### Roles and rights



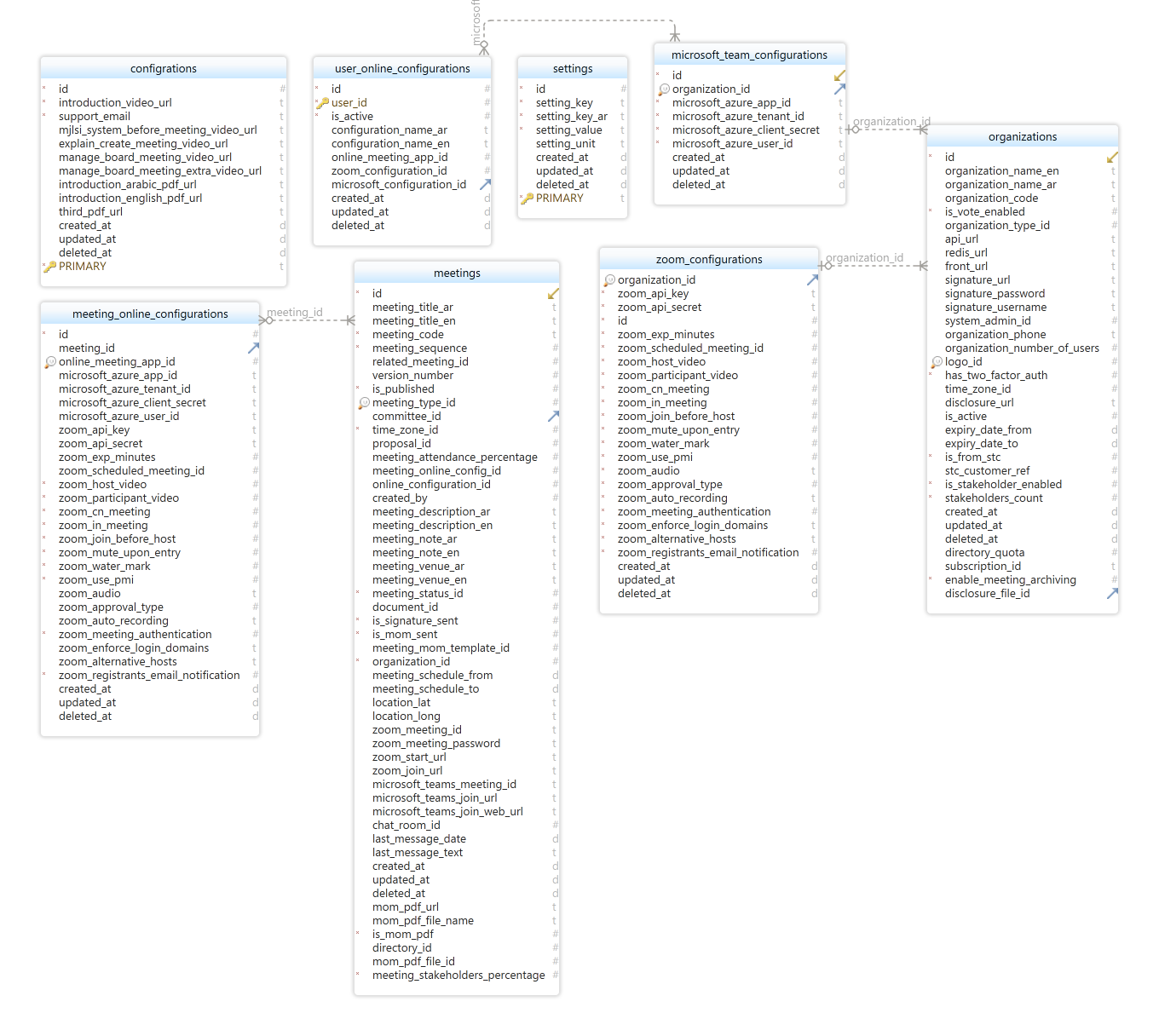
### Tasks



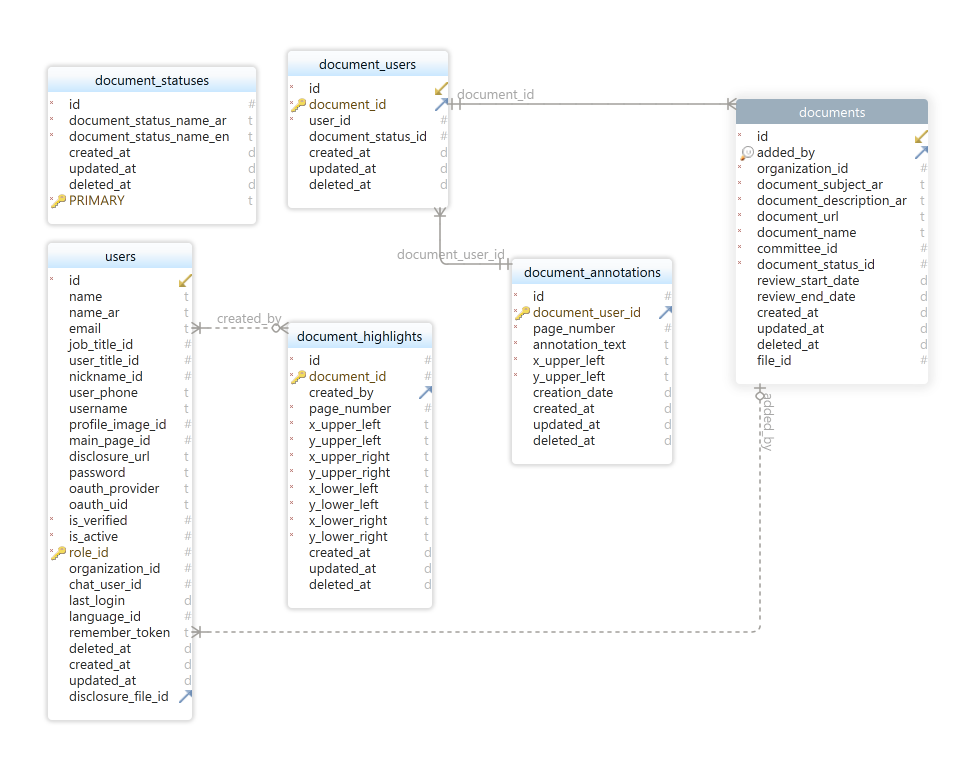
### Users



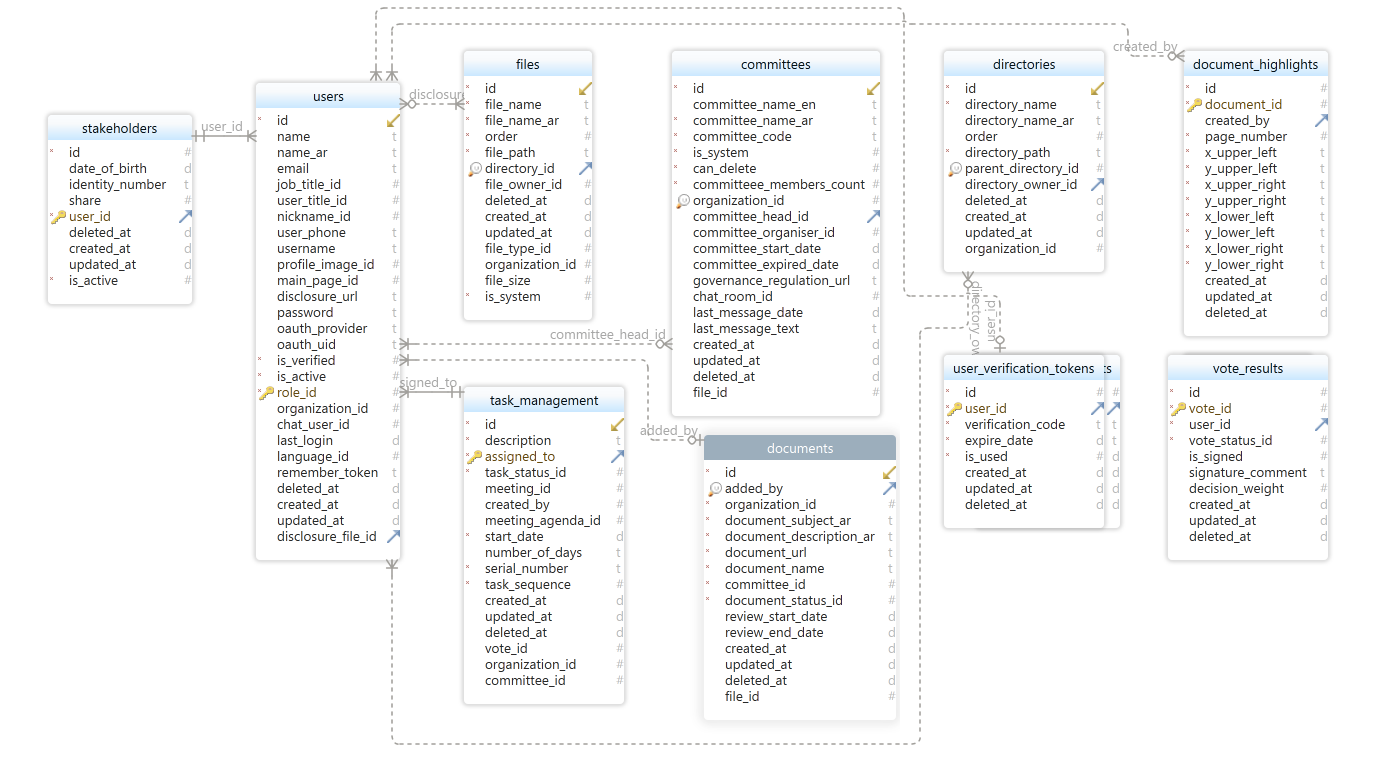
### Configurations and settings



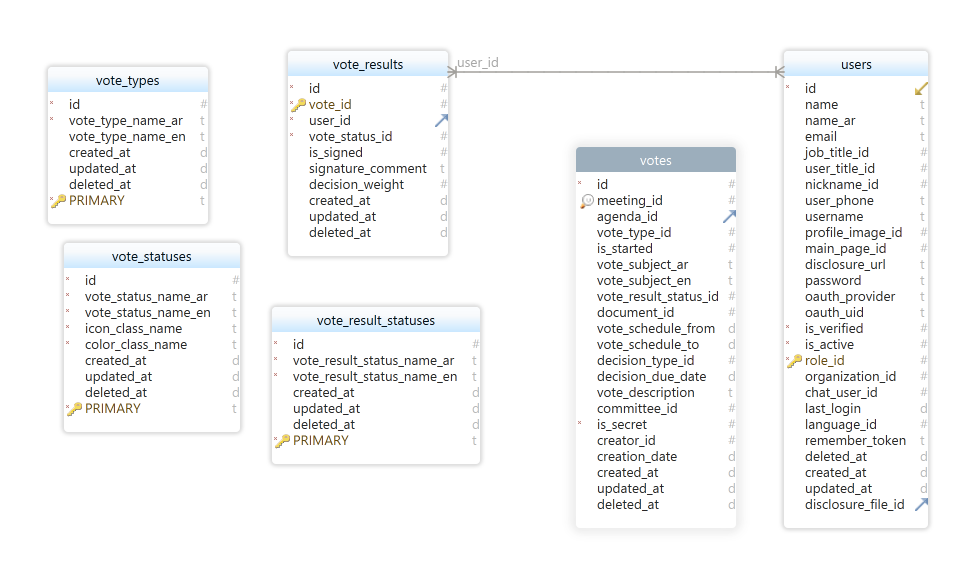
### Documents



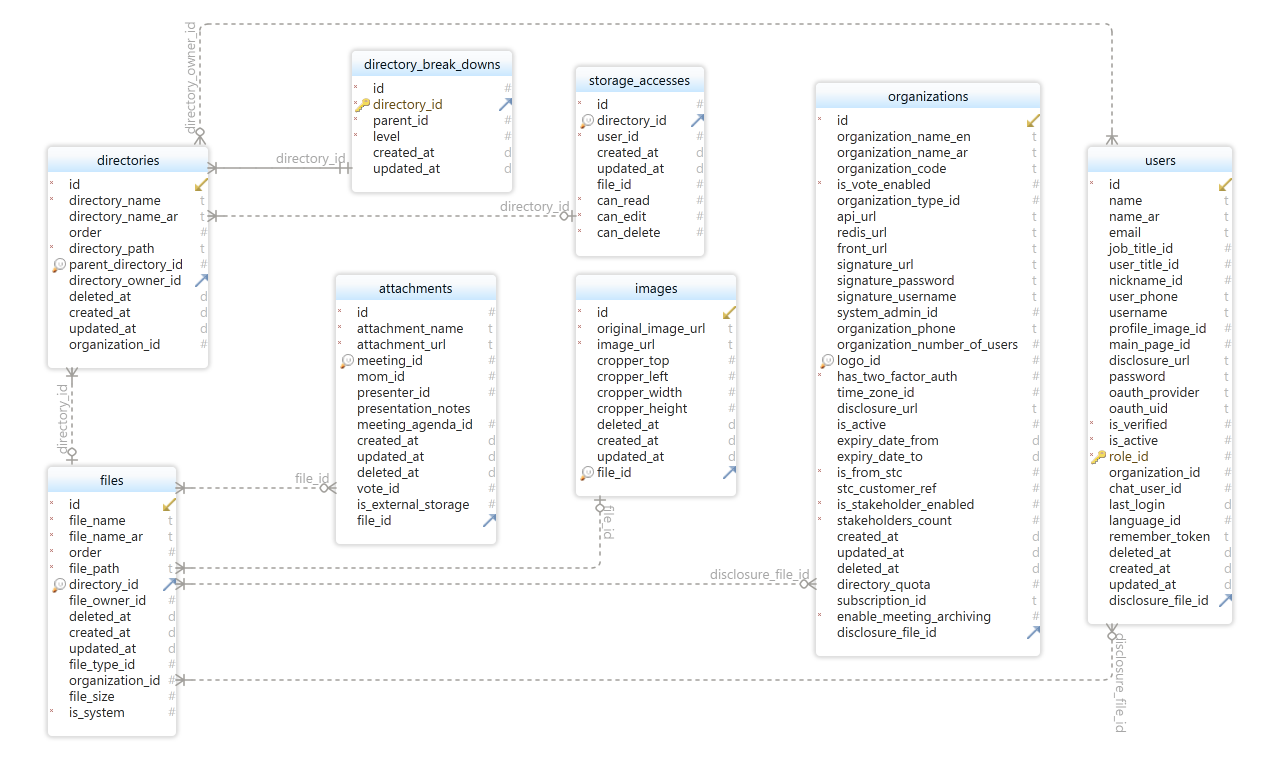
### Stakeholders



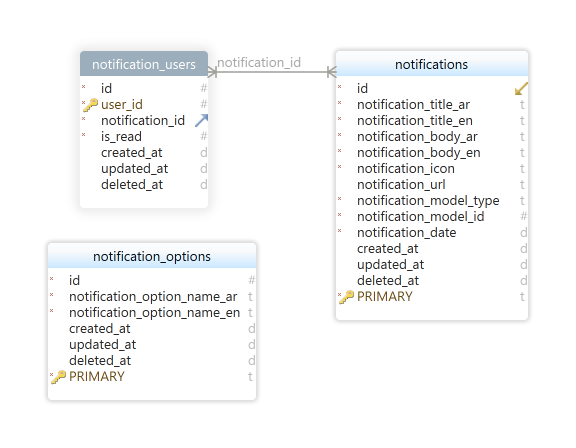
### Votes



### Directories



### Notification



# Development Setup

## Prerequisites

* You must have Visual Studio installed
* You must have IIS installed
* You must have MySQL and Phpmyadmin installed.
* You must have NodeJS installed

## Installation

1. Unzip the code folder
2. Open visual studio
3. Open solution
4. Install dependencies package
5. Configure the connection string and working folders (check configurations section)
6. Run database migration to generate the database
7. In FrontEnd folder Install NPM packages

# System Configurations

The following configurations show the backend configurations that used to set some default settings, or adjustable settings like: connectionstrings, pagesize, SMTP settings etc..

To modify these settings you can find it in the appsettings.json inside the WebAPI project.

| **Logging section**  Configuring the error log | "Logging": {  "IncludeScopes": false,  "LogLevel": {  "Default": "Debug",  "System": "Information",  "Microsoft": "Information"  }  } |
| --- | --- |
| **ConnectionStrings**  Setting the connection string of the database | "ConnectionStrings": {  "DefaultConnection": "server=localhost;port=3306;Database=xxx;SslMode=none;UserId=xxx;Pwd=xxx;CharSet=utf8"  } |
| **PageSize** Setting the default page size for all system grids | "PageSize": 15 |
| **DigitCount**  Setting the default digit count in the system | "DigitCount": "3" |
| **Delimiter** Setting the default delimiter | Delimeter": "-" |
| **DateFormat**  Default date format, user still can modify this | "DateFormat": "dd/MM/yyyy" |
| **Time Format**  Default time format | "TimeFormat": "hh : mm", |
| **Frontend URL** Setting the frontend url | "BaseURL": "/app/#/" |
| **SMTP**  Email configuration | "Smtp": {  "Host": "smtp.sendgrid.net",  "Port": "587",  "Username": "xxxx",  "Password": "xxxx",  "From": "info@xxx.xx",  "DisplayName": "xxxx"  } |

# Deployment

The system is running on the following configuration

## Hardware Configuration

Virtual host provide by DigitalOcean cloud provider

* 8 GB RAM
* 160 GB SSD
* 4 CPUs
* 5 TB Transfer

## Software configuration

* Ubuntu 20.04
* MySQL 8.0