# TAD Micro-services Documentation

## Assumptions

We are looking to migrate BEEP to a microservices architecture.

The specifications are the same as described during all previous iterations of BEEP.

I will answer 10 questions:

- Via the redaction of the TAD (components architecture, sequences diagram, deployment architecture, etc.)
- Via the production of POCs (when mentioned)
- Using draw.io
- Redaction using AsciiDoc

Prohibited technologies:

- Message queue
- CQRS
- Event Sourcing

## Q1: Functional Neighborhoods and Microservices

- Define functionalities in user story format
- Organize into functional neighborhoods
- Propose an architecture diagram

## Q2: Authentication System with OIDC

- Architecture diagram
- Deployment diagram
- Sequence diagrams for user account actions
- Constraints: Use Keycloak, support Polytech LDAP and Google account linking
- POC: Modify Beep, provide deployment guide and source code

## Q3: Inter-Microservices Communication

- Approach explanation

- Sequence diagram of service communication
- POC: HelloWorld microservices communication

## Q4: Authorization System Implementation

- Define authorization service
- Technical and functional architecture
- Sequence diagrams for key actions
- Managing permissions at various levels

## Q5: Logs and Query Tracing

- Define system and technical components
- Deployment diagram
- Sequence diagram for query tracing
- Security-based logs for SOC integration

## Q6: Production-Ready System

- Data security, backup, and restore
- Observability and service supervision
- High availability and continuity plans
- Detailed diagrams

## Q7: Infrastructure Security NO

- Mobile application integration (BFF model)
- Security zones and network separation (DMZ, etc.)
- Cryptography, certificates, and protocol choices

## Q8: Search Engine Integration

- Functional and technical proposal
- Full-text indexing approach
- UI mockup of the search feature
- Technical stack and indexing sequence diagram

## Q9: Platform Security Management NO

- Security analysis for network and microservices
- Communication security between microservices
- Security proposal with sequence diagrams
- POC: Securing HelloWorld microservices

## Q10: UI Application Integration

- Microservice integration for a seamless UX
- Example: Video streaming integration
- Necessary diagrams
- POC: Replacing Beep conference app with microservices-based QCM

## Timeline

- **3 February**: Goals announcement
- **5 March**: Draft report submission (30-50% answers, diagrams, and images)
- **30 March**: Complete V0 file (100% answers, 80% diagrams, 50-70% POCs)
- **30 May 23:42**: Final submission of Git repository containing:
- AsciiDoc file
- Images (PNG + Draw.io sources)
- Production URLs for POCs and repositories

## Goals Announcement

The goal is to implement a microservices architecture for the application.

- The monolith is too big and needs to be split.
- New technical layers are required for maintainability and smooth operation.
- Choose 8 out of the 10 questions (bonus for covering all).
- **Mandatory questions**: 1, 2, 3, 4, 5, and 6.

## Additional Considerations

- DAT vs. DAA complementarity discussion
- Documentation iteration process: initial deployment, real-world constraints, on-premise vs. cloud iterations

- Levels of DAT documentation (light, advanced, very detailed)

This is the documentation for TAD Micro-services version 1.0.0.

# Functional Neighborhoods

## User Authentication

- As a user, I want to sign up for an account so that I can access the application.
- As a user, I want to reset my password so that I can regain access to my account.
- As a user, I want to verify my email address so that I can secure my account.
- As a user, I want to log in to my account so that I can use the application.
- As a user, I want to log out of my account so that I can secure my session.
- As a user, I want to use multi-factor authentication so that I can enhance the security of my account.
- As a user, I want to use a QR code to log in so that I can easily access my account.

## User Profile Management

- As a user, I want to update my profile information so that it reflects my current details.

## Server Management

- As a user, I want to create a new server so that I can have a space for my community.
- As a user, I want to join an exPropose an architecture diagramisting server so that I can participate in a community.
- As a user, I want to leave a server so that I can manage my participation.

## Channel Management

- As a server admin, I want to create channels so that users can have organized discussions.
- As a server admin, I want to delete channels so that I can manage the server's structure.
- As a user, I want to join channels so that I can participate in discussions.

## Roles and Permissions

- As a server admin, I want to assign roles to users so that I can manage their permissions.
- As a server admin, I want to create custom roles so that I can define specific permissions.
- As a server admin, I want to manage permissions at the channel level so that I can control access to discussions.

## Friends

- As a user, I want to add friends so that I can easily communicate with them.
- As a user, I want to remove friends so that I can manage my contacts.
- As a user, I want to accept friend requests so that I can connect with others.

## Messaging

- As a user, I want to send messages in a channel so that I can communicate with others.
- As a user, I want to receive messages in a channel so that I can read others' communications.
- As a user, I want to send direct messages to another user so that I can have private conversations.

## Notifications

- As a user, I want to receive notifications for mentions so that I don’t miss important messages.
- As a user, I want to receive notifications for new messages in channels I follow.

## Voice and Video Communication

- As a user, I want to join voice channels so that I can communicate with others in real-time.
- As a user, I want to start video calls so that I can have face-to-face conversations.
- As a user, I want to share my screen during video calls so that I can collaborate with others.
- As a user, I want to leave voice channels and video calls so that I can end my participation.

## Text Channel Integration

- As a user, I want to be able to create a webhook so that I can integrate text channels with external services.
- As a user, I want to send embeds in text channels so that I can share rich content.

## Internationalization

- As a user, I want to use the application in multiple languages so that I can understand the content better.

---------------------------------------------------

Here’s a structured way to categorize them into functional neighborhoods:

# User Management

## User Authentication:

- Sign up for an account
- Reset password
- Verify email address
- Log in to account
- Log out of account
- Use multi-factor authentication
- Use QR code to log in

## User Profile Management

- Update profile information

## Friends Management

- Add friends
- Remove friends
- Accept friend requests

# Community and Server Management

## Server Management

- Create a new server
- Join an existing server
- Leave a server

## Channel Management

- Create channels (admin)
- Delete channels (admin)
- Join channels (user)

# Communication

### Messaging

- Send messages in a channel
- Receive messages in a channel
- Send direct messages to another user

### Voice and Video Communication

- Join voice channels
- Start video calls
- Share screen during video calls
- Leave voice channels and video calls

## Notifications

### Notification Management

- Receive notifications for mentions
- Receive notifications for new messages in followed channels

## Roles and Permissions

### Roles and Permissions Management

- Assign roles to users (admin)
- Create custom roles (admin)
- Manage permissions at the channel level (admin)

## Integration and Customization

### Text Channel Integration

- Create a webhook
- Send embeds in text channels

### Internationalization

- Use the application in multiple languages

# Functional Neighborhoods and Microservices

## User Management

### User Authentication Service

- Sign up for an account
- Reset password
- Verify email address
- Log in to account
- Log out of account
- Use multi-factor authentication
- Use QR code to log in

### User Profile Service

- Update profile information

### Friends Management Service

- Add friends
- Remove friends
- Accept friend requests

## Community and Server Management

### Server Management Service

- Create a new server
- Join an existing server
- Leave a server

### Channel Management Service

- Create channels (admin)
- Delete channels (admin)

- Join channels (user)

## Communication

### Messaging Service

- Send messages in a channel
- Receive messages in a channel
- Send direct messages to another user

### Voice and Video Service

- Join voice channels
- Start video calls
- Share screen during video calls
- Leave voice channels and video calls

## Notifications

### Notification Service

- Receive notifications for mentions
- Receive notifications for new messages in followed channels

## Roles and Permissions

### Roles and Permissions Service

- Assign roles to users (admin)
- Create custom roles (admin)
- Manage permissions at the channel level (admin)

## Integration and Customization

### Text Channel Integration Service

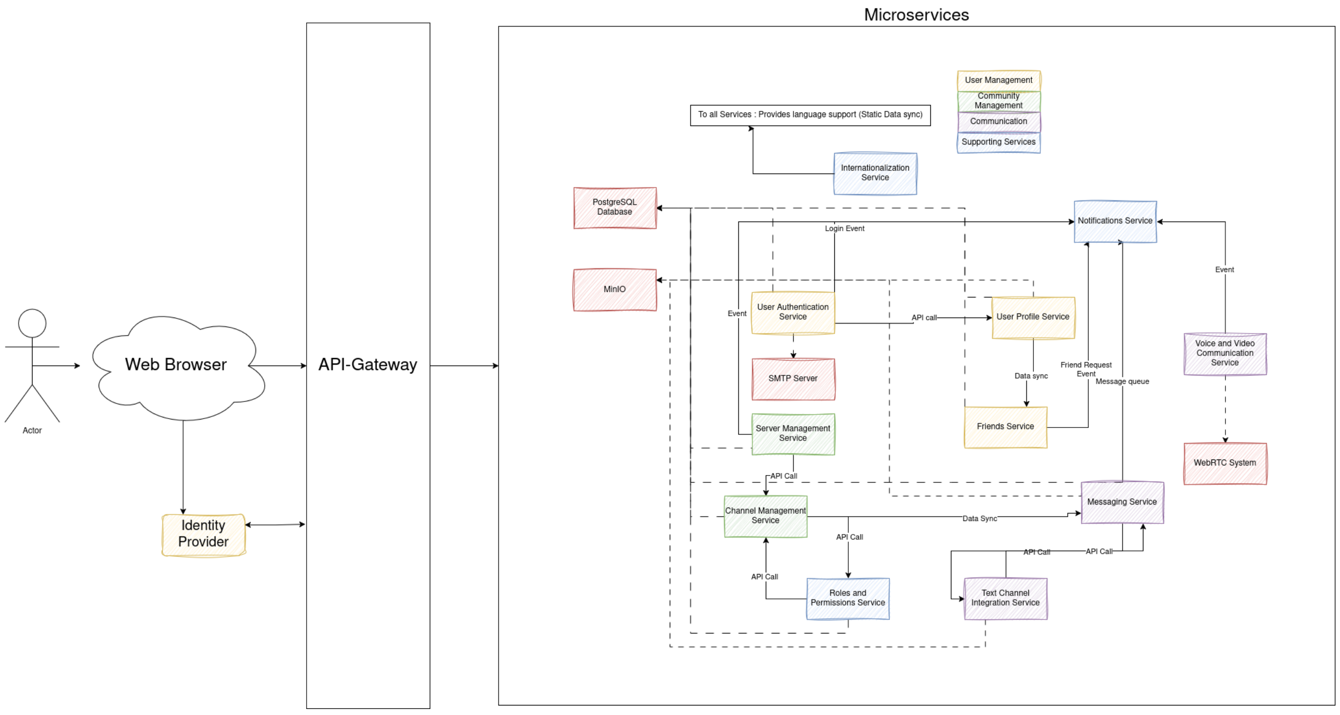
- Create a webhook
- Send embeds in text channels

## Internationalization Service

- Use the application in multiple languages

## Proposed Architecture Diagram

The architecture diagram represents these microservices as distinct components, each responsible for a specific set of functionalities. Each service communicates with others through well-defined APIs, ensuring loose coupling and high cohesion. This architecture promotes scalability, maintainability, and independent deployment of each service.



## Key Considerations

- **Single Responsibility Principle**: Each microservice should focus on a single responsibility, ensuring that it does one thing well.
- **Cohesion**: Related functionalities should be grouped together within the same microservice.
- **Loosely Coupled**: Minimize dependencies between microservices to reduce the impact of changes.
- **Scalability**: Ensure that each microservice can scale independently based on demand.
- **Resilience**: Design the system to handle failures in individual microservices without affecting the entire application. == Q2: How do I manage the authentication system with an OIDC?

## What is OpenID?

OpenID is a protocol that allows for the verification of user identities based on authentication performed by an Authorization Server and obtains user profile information in an interoperable and REST-like manner.

## How does OpenID Connect work?

OpenID Connect enables an Internet identity ecosystem. It follows these steps:

1. The User navigates to a website via a web browser.
2. The User clicks sign-in and types their username and password.
3. The RP (Client) sends an authentication request to the OpenID Provider (OP).
4. The OP authenticates the User and obtains authorization.
5. The OP sends an Identity Token to the RP and usually an Access Token.
6. The RP can send a request with the Access Token to the User device.
7. The UserInfo Endpoint returns Claims about the End-User.

## What are the benefits of OpenID Connect?

- Simplified User Authentication
- Secure User Authentication
- Single Sign-On (SSO)
- User Profile Information
- Standardized
- Interoperable
- REST-like

## What are the components of OpenID Connect?

- **End-User**: The person who uses the RP.
- **RP (Relying Party)**: The application that wants to authenticate the End-User.
- **OP (OpenID Provider)**: The service that authenticates the End-User and provides Claims to the RP.
- **Authorization Server**: The server that authenticates the End-User and issues Access Tokens.
- **Identity Token**: A JWT that contains information about the End-User.

# What is Keycloak?

Keycloak is an open-source Identity and Access Management solution that makes it easy to secure applications and services with little to no code.

https://www.keycloak.org/server/containers

## How does Keycloak work?

Keycloak works by providing an authentication server that can be used to secure applications and services. It supports OpenID Connect, OAuth 2.0, and SAML 2.0 protocols, making it easy to integrate with existing systems.

Keycloak provides a wide range of features, including:

- User Authentication
- User Registration
- User Management
- Role-Based Access Control
- Single Sign-On
- Multi-Factor Authentication
- Social Login
- User Sessions
- Account Management
- Admin Console
- REST API
- Scalability

## What are the main components of Keycloak?

Keycloak has two main components, including:

- **Keycloak Server**: The server that provides the authentication and authorization services. It includes the API and a graphical interface.
- **Keycloak Client**: The client that interacts with the Keycloak server to authenticate users and obtain tokens.

# How to integrate Keycloak with an OIDC?

To integrate Keycloak with an OIDC, we need to follow these steps:

1. **Create a Realm**: Create a new realm in Keycloak to manage the authentication and authorization settings for the application.
2. **Create a Client**: Create a new client in the realm to represent the application.
3. **Configure the Client**: Configure the client settings, such as the client ID, client secret, and redirect URIs.
4. **Integrate the Client**: Integrate the client with the application by adding the necessary code to handle the authentication flow.

5. **Test the Integration**: Test the integration by logging in to the application and verifying that the authentication works as expected.

## How to secure an application with Keycloak?

To secure an application with Keycloak, we need to follow these steps:

1. **Add Keycloak Dependencies**: Add the Keycloak dependencies to the application, such as the Keycloak adapter.
2. **Configure Keycloak Adapter**: Configure the Keycloak adapter in the application to connect to the Keycloak server.
3. **Secure Endpoints**: Secure the application endpoints by adding Keycloak security constraints.
4. **Add Keycloak Login Page**: Add the Keycloak login page to the application to handle the authentication flow.

# Architecture Diagram

## Identifying the components:

- **Client (Frontend)**: User interface (web or mobile).
- **Application Server (Backend)**: Business logic and API.
- **Keycloak Server**: Authentication and authorization server.
- **Database**: Storage for user and application data.
- **External Services**: Google OAuth, Polytech LDAP.

## Define the interactions:

- Authentication flow between the client, application server, and Keycloak.
- Integration with Google OAuth and Polytech LDAP for authentication.
- Communication between the application server and the database.

# Deployment Diagram

## Identifying the infrastructure:

- Physical or virtual servers.
- Cloud services (AWS, Azure, GCP).
- Containers (Docker, Kubernetes).

## Define the topology:

- Placement of components on servers/containers.
- Network configuration (load balancers, firewalls).

## Sequence Diagram

User creates Beep account (vanilla)

```
User                Beep Client                User Auth Service          Database
 |                       |                          |                          |
 |---(1) Submit Reg Form->|                          |                          |
 |                       |---(2) Create Account---->|                          |
 |                       |                          |---(3) Save User Data------>|
 |                       |                          |<--(4) Confirm Data Saved---|
 |                       |<--(5) Account Created----|                          |
 |<--(6) Success Message-|                          |                          |
```

User creates Beep account (via Polytech account)

```
User                Beep Client                Keycloak (OIDC)            Polytech
OIDC
 |                       |                          |                          |
 |---(1) Select Polytech->|                          |                          |
 |                       |---(2) Redirect Login--->|                          |
 |                       |                          |---(3) Polytech Login Page->|
 |---(4) Enter Credentials|                          |                          |
 |                       |                          |<--(5) Authenticate User----|
 |                       |<--(6) Auth Token---------|                          |
 |                       |---(7) Create Account---->|                          |
 |                       |                          |---(8) Save User Data------>|
 |                       |                          |<--(9) Confirm Data Saved---|
 |                       |<--(10) Account Created---|                          |
 |<--(11) Success Message|                          |                          |
```

User creates Beep account (via Google account)

```
User                Beep Client                Keycloak (OIDC)            Google
OIDC
 |                       |                          |                          |
 |---(1) Select Google->|                          |                          |
 |                       |---(2) Redirect Login--->|                          |
 |                       |                          |---(3) Google Login Page--->|
 |---(4) Enter Credentials|                          |                          |
 |                       |                          |<--(5) Authenticate User----|
 |                       |<--(6) Auth Token---------|                          |
 |                       |---(7) Create Account---->|                             |
 |                       |                          |---(8) Save User Data------>|
 |                       |                          |<--(9) Confirm Data Saved---|
 |                       |<--(10) Account Created---|                             |
 |<--(11) Success Message|                          |                             |
```

User logs in (vanilla)

```
User                Beep Client                User Auth Service           Database
 |                       |                          |                             |
 |---(1) Enter Credentials->|                          |                             |
 |                       |---(2) Login Request----->|                             |
 |                       |                          |---(3) Verify Credentials-->|
 |                       |                          |<--(4) Confirm Credentials--|
 |                       |<--(5) Login Success------|                             |
 |<--(6) Display Dashboard|                          |                             |
```

User logs in (Polytech)

```
User                Beep Client                Keycloak (OIDC)              Polytech
OIDC
 |                       |                          |                             |
 |---(1) Select Polytech->|                          |                             |
 |                       |---(2) Redirect Login--->|                             |
 |                       |                          |---(3) Polytech Login Page->|
 |---(4) Enter Credentials|                          |                             |
 |                       |                          |<--(5) Authenticate User----|
 |                       |<--(6) Auth Token---------|                             |
 |                       |---(7) Validate Token---->|                             |
 |                       |<--(8) Login Success------|                             |
 |<--(9) Display Dashboard|                          |                             |
```

User associates their Google account with their Beep account

```
User                Beep Client                Keycloak (OIDC)              Google
OIDC
 |                       |                          |                             |
 |---(1) Initiate Linking->|                          |                             |
 |                       |---(2) Redirect Login--->|                             |
 |                       |                          |---(3) Google Login Page--->|
 |---(4) Enter Credentials|                          |                             |
 |                       |                          |<--(5) Authenticate User----|
 |                       |<--(6) Auth Token---------|                             |
 |                       |---(7) Link Google Account->|                             |
 |                       |                          |---(8) Save Linked Info---->|
 |                       |                          |<--(9) Confirm Data Saved---|
 |                       |<--(10) Linking Success---|                             |
    |<--(11) Success Message|                           |                              |
```

# POC (Proof of Concept)

# OpenID Connect (OIDC)

OpenID Connect (OIDC) is an authentication protocol built on top of the OAuth 2.0 framework. It provides a standardized method for verifying the identity of users and obtaining basic profile information in a secure and interoperable manner.

## Key Features

- **Identity Verification**: OIDC allows applications to verify the identity of users based on authentication performed by an Authorization Server. This is done using JSON Web Tokens (JWTs), which contain claims about the user's identity.
- **Interoperability**: OIDC is designed to work across different platforms and devices, making it a versatile solution for modern applications. It supports a wide range of clients, including web applications, mobile apps, and JavaScript applications.
- **Security and Privacy**: OIDC enhances security by removing the need for applications to manage user passwords. Instead, it relies on tokens issued by an Identity Provider (IdP), which can be easily revoked or refreshed.
- **Single Sign-On (SSO)**: OIDC supports SSO, allowing users to authenticate once and gain access to multiple applications without needing to log in again. This improves user experience and reduces the administrative burden of managing multiple credentials.
- **Standardization**: OIDC is a standard protocol maintained by the OpenID Foundation, ensuring compatibility and ease of integration with various systems and services.

## How It Works

1. **Authentication Request**: A client application (Relying Party) initiates an authentication request to the OpenID Provider (OP).
2. **User Authentication**: The OP authenticates the user and obtains their consent to share specific information with the client application.
3. **Token Issuance**: Upon successful authentication, the OP issues tokens to the client application. These tokens include an ID token, which contains user identity information, and optionally an access token for API access.
4. **Identity Verification**: The client application uses the ID token to verify the user's identity and can use the access token to access protected resources on behalf of the user.

## Benefits

- **Simplified Integration**: OIDC simplifies the integration of authentication and authorization in applications, reducing development time and effort.

- **Enhanced Security**: By relying on tokens instead of passwords, OIDC reduces the risk of credential-based data breaches.
- **User Consent**: OIDC includes built-in user consent mechanisms, ensuring that users have control over their personal data.

## Conclusion

OpenID Connect is a powerful and flexible authentication protocol that enhances security, interoperability, and user experience in modern applications. Its adoption by major technology companies and its standardization make it a reliable choice for identity management in various scenarios. = Q3. Inter microservices communicationa

# Communication Protocols

- **HTTP/HTTPS**: The most common protocol for inter-microservice communication. It is widely supported, easy to use, and can be secured using HTTPS.

# Orchestration Approach

- **API Gateway**: Acts as a single entry point for all client requests. It routes requests to the appropriate microservice and handles tasks like load balancing, authentication, and rate limiting.
- **Load Balancing**: Distributes incoming traffic across multiple instances of a microservice to ensure no single instance is overwhelmed.
- **Centralized Logging and Monitoring**: Tools like the LGTM suite can helps monitor the health and performance of microservices.

# Sequence Diagram

Below is a textual representation of a sequence diagram for user authentication involving multiple microservices:

```
User                API Gateway               Auth Service                Profile
Service
 |                      |                         |                            |
 |---(1) Login Request-->|                         |                            |
 |                      |---(2) Forward Request-->|                            |
 |                      |                         |---(3) Validate Credentials |
 |                      |                         |<--(4) Return Token---------|
 |                      |<--(5) Return Token-------|                            |
 |<--(6) Return Token----|                         |                            |
 |                      |                         |                            |
 |---(7) Profile Request->|                         |                            |
 |                       |---(8) Forward Request-->|                           |
 |                       |                          |---(9) Fetch Profile Info-->|
 |                       |                          |<--(10) Return Profile Info-|
 |                       |<--(11) Return Profile Info|                            |
 |<--(12) Return Profile Info                       |                            |
```

# Proof of Concept (POC)

To demonstrate inter-microservice communication, we can create a simple "Hello World" example using HTTP and AdonisJS.

## Microservice 1: Greeting Service

### Create a new AdonisJS project:

```
npm init adonis-ts-app@latest greeting-service
cd greeting-service
```

### Install necessary dependencies:

```
npm install @adonisjs/core @adonisjs/lucid
```

### Create a route and controller to handle the greeting:

**start/routes.ts**

```
import Route from '@ioc:Adonis/Core/Route'

Route.get('/greet', 'HelloWorldController.index')
Route.get('/', 'HelloWorldController.index')
```

**app/Controllers/Http/GreetingsController.ts**

```
node ace make:controller HelloWorldController
```

```
import { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'

export default class HelloWorldController {
  public async index({ response }: HttpContextContract) {
    return response.json({ message: 'Hello World' })
  }
}
```

### Start the Greeting Service:

```
node ace serve --watch
```

## Microservice 2: Client Service

1. Create another AdonisJS project:

```
npm init adonis-ts-app@latest client-service
cd client-service
```

### Install necessary dependencies:

```
npm install @adonisjs/core @adonisjs/lucid axios
```

### Create a route and controller to handle the client request:

**start/routes.ts**

```
import Route from '@ioc:Adonis/Core/Route'

Route.get('/', async ({ view }) => {
  return view.render('welcome')})

Route.get('/hello', 'ClientController.hello')
```

**app/Controllers/Http/ClientController.ts**

```
node ace make:controller ClientController
```

```
import { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'
import axios from 'axios'

export default class ClientController {
  public async hello({ response }: HttpContextContract) {
    try {
      const res = await axios.get('http://localhost:3334/')
      const greeting = res.data.message
      return response.json({ message: `Client Service says: ${greeting}` })
    } catch (error) {
      return response.internalServerError({ message: 'Error fetching greeting' })
    }
  }
}
```

### Change the port in the `.env` file to 3334:

```
PORT=3334
```

### Start the Client Service:

```
node ace serve --watch
```

You can have some issues because of an -s in the name of the controllers. If you are still having troubles, make sure to check the official AdonisJS documentation for more information.

## Running the POC

1. Start the Greeting Service on port 3334.
2. Start the Client Service on port 3333.
3. Access `http://localhost:3333/hello` to see the communication between the two services.

# Authorization System Overview

An authorization system determines whether a user has permission to perform specific actions or access certain resources within an application. It is a critical component of security, ensuring that users can only access data and functionality appropriate to their roles and permissions.

## Authorization Service Definition

An authorization service is responsible for managing and enforcing access control policies. It verifies user permissions based on roles, groups, or other attributes and grants or denies access to resources accordingly.

## Technical and Functional Architecture

### Technical Architecture

1. **Microservices Architecture**:
    - **Authorization Service**: Centralized service responsible for managing permissions and

    roles.
    - **User Service**: Manages user data and authentication.
    - **Resource Services**: Individual microservices for servers, categories and channels, each communicating with the Authorization Service to enforce access control.
2. **Database**:
    - Use a relational database (e.g., PostgreSQL) to store user roles, permissions, and access control lists (ACLs).
3. **API Gateway**:
    - Acts as a single entry point for all client requests, routing them to the appropriate microservice and handling authentication and authorization checks.
4. **JWT (JSON Web Tokens)**:
    - Used for securely transmitting information between parties as a JSON object. It can be signed using a secret or a public/private key pair.
5. **OAuth 2.0**:
    - An authorization framework that enables applications to obtain limited access to user accounts on an HTTP service.
6. **Caching**:
    - Use a caching layer (e.g., Redis) to store frequently accessed permissions data to improve performance.

## Functional Architecture

1. **Role-Based Access Control (RBAC)**:
    - Define roles (e.g., admin, moderator, user) with associated permissions.
    - Assign roles to users or groups of users.
2. **Access Control Lists (ACLs)**:
    - Define permissions at a granular level (e.g., server, category, channel).
    - Allow or deny access based on user roles and ACLs.
3. **Global Permissions**:
    - Define global permissions for platform administrators, allowing them to manage all resources across the platform.

# Sequence Diagrams

## User Login and Authorization Check

```
User                API Gateway                 User Service                    Auth Service
Resource Service
```

```
 |                       |                          |                             |
|
 |---(1) Login Request-->|                          |                             |
|
 |                       |---(2) Authenticate User->|                             |
|
 |                       |                          |---(3) Verify Credentials-->|
|
 |                       |                          |                             |---
(4) Fetch User Permissions|
 |                       |                          |<--(5) Return JWT Token-----|
|
 |                       |<--(6) Return JWT Token---|                             |
|
 |<--(7) Return JWT Token|                          |                             |
|
 |                       |                          |                             |
|
 |---(8) Access Resource Request (with JWT)-------->|                          |
|                              |
 |                       |---(9) Validate Permissions->                          |
|                              |
 |                       |                          |                             |---
(10) Fetch User Permissions|
 |                       |                          |                             |<--
(11) Permission Validation Result|
 |                       |<--(12) Permission Validation Result                  |
|                              |
 |                       |---(13) Forward Request (if authorized)--------------->|
|                              |
 |                       |                          |                             |
|<--(14) Return Resource Data|
 |<--(15) Return Resource Data                       |                          |
|                              |
```

## Managing Permissions by Server, Category, and Channel

```
Admin              API Gateway               Auth Service
 |                       |                          |
 |---(1) Update Permissions Request-->|                          |
 |                       |---(2) Update Permissions->|
 |                       |                          |---(3) Store Updated Permissions|
 |                       |                          |<--(4) Confirmation-----------|
 |                       |<--(5) Confirmation-------|                          |
 |<--(6) Confirmation----|                          |                          |
```

## Global Permissions for Platform Administrators

```
Admin               API Gateway               Auth Service                Resource
Service
 |                       |                          |                             |
 |---(1) Access Global Resource Request-->|                          |
|
 |                       |---(2) Validate Global Permissions->
|                             |
 |                       |                          |---(3) Fetch Global Permissions|
|
 |                       |                          |<--(4) Permission Validation
Result|                             |
 |                       |<--(5) Permission Validation Result                   |
|
 |                       |---(6) Forward Request (if authorized)--------------->|
|
 |                       |                          |                             |<--
(7) Return Resource Data|
 |<--(8) Return Resource Data                      |                          |
|
```

# Recommended Technologies

1. **Backend Framework**:
   - AdonisJS for building microservices in Node.js.
2. **Database**:
   - PostgreSQL for storing user data, roles, and permissions.
3. **Authentication and Authorization**:
   - OAuth 2.0 and JWT for secure authentication and authorization. Use of Keycloak for OIDC support.
4. **API Gateway**:
   - NGINX for managing API requests and enforcing security policies.
5. **Caching**:
   - Redis for caching frequently accessed permissions data.
6. **Monitoring and Logging**:
   - The LGTM stack for monitoring and logging.

# Logging and Query Tracing System

# System and Technical Components

To effectively trace logs and queries, implement the following components:

1. **Logging Framework**: Use a logging framework like winston in Adonis to log events, errors, and other relevant information.
2. **Centralized Log Management**: Use Loki with Grafana to aggregate and visualize logs from different services. Loki is a horizontally scalable, highly available, multi-tenant log aggregation system.
3. **Distributed Tracing**: Implement distributed tracing with Tempo to trace requests as they flow through different services. Tempo is a highly scalable and efficient tracing backend.
4. **API Gateway**: Acts as an entry point for all client requests and can log request/response data.
5. **Monitoring Tools**: Use Grafana with Mimir for monitoring and visualizing system metrics. Mimir is a scalable, multi-tenant time series database that integrates seamlessly with Grafana.
6. **Security Information and Event Management (SIEM)**: Integrate with a SIEM system for security-based logs and incident response.

# Deployment Diagram (change to adapt with OIDC)

```
                                 +----------------------+
                                 |     API Gateway      |
                                 +----------+-----------+
                                            |
                                            |
                                            v
+------------------+       +----------------+----------------+
+------------------+
|                  |       |                |                |       |
|
|   User Service   |<----->|  Auth Service  |<-------------->|  Resource Service    |
|                  |       |                |                |       |
|
+------------------+       +----------------+----------------+
+------------------+
            ^                         ^                   ^                   ^
            |                         |                   |                   |
            v                         v                   v                   v
+------------------+       +----------------+----------------+
+------------------+
|   Logging        |<----->|  Centralized   |<-------------->|  Distributed Tracing  |
|   Framework      |       |  Log Management|                |  |
+------------------+       +----------------+----------------+
+------------------+
            ^                         ^
```

```
          |                        |
          v                        v
+-----------------+      +---------------+---------------+
|  Monitoring     |<----->|  Security Information and Event  |
|  Tools (e.g.,   |      |  Management (SIEM) System       |
|  Prometheus,    |      |                                 |
|  Grafana)       |      +---------------+---------------+
+-----------------+
```

## Sequence Diagram for a Query (Change for OIDC)

Below is a sequence diagram for a user login request, illustrating how the system works with logging and tracing:

```
User                API Gateway             User Service              Auth Service
Log Management              Distributed Tracing
 |                      |                          |                            |
|                       |
 |---(1) Login Request-->|                          |                            |
|                       |
 |                      |---(2) Log Request-------->|                            |
|                       |
 |                      |---(3) Authenticate User->|                            |
|                       |
 |                      |                          |---(4) Verify Credentials-->|
|                       |
 |                      |                          |                            |---
(5) Log Credential Check->|                            |
 |                      |                          |                            |---
(6) Trace Request-------->|                            |
 |                      |                          |<--(7) Return JWT Token-----|
|                       |
 |                      |<--(8) Return JWT Token---|                            |
|                       |
 |<--(9) Return JWT Token|                          |                            |
|                       |
 |                      |---(10) Log Response------>|                            |
|                       |
 |                      |                          |                            |
|---(11) Trace Response------>|
```

## Security-Based Logs for Managed SOC

To integrate with a managed Security Operations Center (SOC), consider the following needs:

1. **Comprehensive Logging**
    - **Objective**: Capture all security-relevant events to provide a complete audit trail.
    - **Implementation**:
    - **Authentication Attempts**: Log both successful and failed login attempts, including details like user ID, timestamp, and IP address.
    - **Authorization Failures**: Record instances where users attempt to access resources they are not authorized to view.
    - **Data Access Events**: Log access to sensitive data, including who accessed it, when, and from where.
    - **System Changes**: Track changes to system configurations, user permissions, and other critical settings.
2. **Real-time Monitoring**
    - **Objective**: Detect and respond to security incidents as they occur.
    - **Implementation**:
    - **Monitoring Tools**: Use tools that provide real-time visibility into security events and anomalies.
    - **Alerting**: Set up alerts for suspicious activities, such as multiple failed login attempts or unusual data access patterns.
    - **Automated Responses**: Implement automated responses to common threats, such as blocking IP addresses after multiple failed login attempts.
3. **Compliance**
    - **Objective**: Ensure that logging practices comply with relevant regulations and standards.
    - **Implementation**:
    - **GDPR**: Ensure that logs containing personal data are handled in compliance with GDPR requirements, including data minimization and anonymization.
4. **Incident Response**
    - **Objective**: Define and implement procedures for responding to security incidents.
    - **Implementation**:
    - **Incident Response Plan**: Develop a plan that outlines the steps to take in response to different types of security incidents.
    - **Log Analysis**: Include procedures for analyzing logs to identify the root cause of incidents.
    - **Forensic Investigation**: Ensure that logs are preserved and available for forensic analysis.
5. **Integration with SIEM**
    - **Objective**: Use a Security Information and Event Management (SIEM) system to correlate logs from various sources and detect potential security threats.
    - **Implementation**:
    - **Log Aggregation**: Collect logs from all relevant sources, including applications, servers, and network devices.

    - **Correlation Rules**: Define rules to correlate events across different logs and identify potential security threats.
    - **Threat Detection**: Use the SIEM system to detect and alert on known threat patterns and anomalies.
6. **Access Control**
    - **Objective**: Implement strict access controls to protect log data from unauthorized access.
    - **Implementation**:
    - **Role-Based Access Control (RBAC)**: Define roles and permissions to control access to log data.
    - **Encryption**: Encrypt log data both at rest and in transit to protect it from unauthorized access.
    - **Audit Access**: Regularly audit access to log data to ensure compliance with access control policies.

## Benefits

- **Operational Visibility**: Comprehensive logging and real-time monitoring provide visibility into security events, enabling proactive threat detection and response.
- **Incident Response**: Defined incident response procedures and integration with SIEM systems enable quick and effective response to security incidents.
- **Access Control**: Strict access controls protect log data from unauthorized access, maintaining the integrity and confidentiality of security-relevant information.

# Data Security, Data Backup, and Restore

## Data Security

- **Objective**: Protect sensitive data from unauthorized access, breaches, and corruption.
- **Implementation**:
- **Encryption**: Use encryption for data at rest and in transit to protect it from unauthorized access.
- **Access Control**: Implement role-based access control (RBAC) to ensure only authorized personnel can access sensitive data.
- **Regular Audits**: Conduct regular security audits and vulnerability assessments to identify and mitigate risks.
- **Compliance**: Ensure compliance with relevant data protection regulations such as GDPR, HIPAA, etc.

## Data Backup and Restore

- **Objective**: Ensure data can be recovered in case of loss or corruption.
- **Implementation**:
- **Regular Backups**: Schedule regular automated backups of critical data.
- **Offsite Storage**: Store backups offsite or in the cloud to protect against physical damage or loss.
- **Restore Procedures**: Develop and test restore procedures to ensure data can be recovered quickly and accurately.
- **Versioning**: Use versioning to keep track of changes and allow for point-in-time recovery.

## Diagram: Data Security and Backup

```
TODO: Add diagram for Data Security and Backup
```

# Observability and Services Supervision

## Observability

- **Objective**: Monitor the health and performance of the system in real-time.
- **Implementation**:
- **Logging**: Implement comprehensive logging for all system components.
- **Metrics**: Collect and monitor key performance metrics.
- **Tracing**: Use distributed tracing to track requests across microservices.
- **Dashboards**: Create dashboards to visualize system health and performance.

## Services Supervision

- **Objective**: Ensure services are running as expected and take corrective actions when they are not.
- **Implementation**:
- **Health Checks**: Implement health checks for each service to monitor their status.
- **Alerting**: Set up alerts for service failures or performance degradation.
- **Automated Recovery**: Implement automated recovery procedures for common failures.
- **Incident Management**: Develop incident management procedures to handle service disruptions.

## Diagram: Observability and Services Supervision

```
TODO: Add diagram for Observability and Services Supervision
```

# Infrastructure High Availability and Continuity Plans

## High Availability

- **Objective**: Ensure the system remains operational and accessible even in the event of failures.
- **Implementation**:
- **Redundancy**: Implement redundant components for critical systems.
- **Load Balancing**: Use load balancers to distribute traffic and prevent overloads.
- **Failover Mechanisms**: Implement automatic failover mechanisms to switch to backup systems in case of failure.
- **Regular Testing**: Regularly test failover and recovery procedures to ensure they work as expected.

## Continuity Plans

- **Objective**: Ensure business operations can continue with minimal disruption in the event of a major incident.
- **Implementation**:
- **Business Impact Analysis (BIA)**: Conduct a BIA to identify critical systems and processes.
- **Recovery Time Objectives (RTO)**: Define RTOs for critical systems.
- **Recovery Point Objectives (RPO)**: Define RPOs for data recovery.
- **Disaster Recovery Plan**: Develop and maintain a disaster recovery plan that includes procedures for restoring systems and data.

## Diagram: High Availability and Continuity Plans

```
TODO: Add diagram for High Availability and Continuity Plans
```

## Target Diagram

```
TODO: Add target diagram
```