# Channel Employees Server

Microservice to manage employees of a channel.

**Note**: This Service uses a Json file named “config.json” as its configuration file.

## Technologies

* Back: NodeJS
* Database: MySQL
* Messaging: Kafka

## Basic Response Model

* All responses are JSON objects
* All responses must have at least these two parameters
  + status: “success” or “failed”
  + message: String
* All responses must include response code (200 for success, non 200 for failures)

## Part 1: Authorization

Use Authorization Client library to provide access to Authorization server.

In this document, wherever mentioned “Authorization” refers to Authorization client library.

/members/bind

Bind a user to a channel.

### Parameters

* user\_id: String
* bid: String
* chid: String
* member\_id: String

### Returns

* <Basic Response>

### Steps

1. Call Authorization.authorize(“channel\_employees/members/bind”, {user\_id, bid, chid, member\_id}) to check authorization
2. Create new record in members table with {bid, chid, member\_id, role=”\_employee”
3. Publish event on Kafka
   1. Topic: “channel\_employees\_employee\_bound”
   2. Content:
      1. user\_id
      2. bid
      3. chid
      4. member\_id
      5. role=”\_employee”
      6. created\_at
4. Return Success

/members/bindAll

Bind a list of users to a channel.

### Parameters

* user\_id: String
* bid: String
* chid: String
* member\_ids: String Array

### Returns

* <Basic Response>

### Steps

1. Call Authorization.authorize(“channel\_employees/members/bind”, {user\_id, bid, chid, member\_ids}) to check authorization
2. For each member\_id in member\_ids create new record in members table with {bid, chid, member\_id, role=”\_employee”
3. Publish event on Kafka
   1. Topic: “channel\_employees\_employee\_group\_bound”
   2. Content:
      1. user\_id
      2. bid
      3. chid
      4. member\_ids
      5. role=”\_employee”
      6. created\_at
4. Return Success

/members/updateRole

Update role of employee.

### Parameters

* user\_id: String
* bid: String
* chid: String
* member\_id: String
* role:String

### Returns

* <Basic Response>

### Steps

1. Call Authorization.authorize(“channel\_employees/members/role:update”, {user\_id, member\_id, role}) to check authorization
2. Update role of record in members table with same {bid, chid, member\_id}
3. Publish event on Kafka
   1. Topic: “channel\_employees\_employee\_role\_updated”
   2. Content:
      1. by: user\_id
      2. bid
      3. chid
      4. member\_id
      5. role
      6. created\_at
4. Return Success

/members/unbind

Unbind a user from a channel.

### Parameters

* user\_id: String
* bid: String
* chid: String
* member\_id: String

### Returns

* <Basic Response>

### Steps

1. Call Authorization.authorize(“channel\_employees/members/unbind”, {user\_id, bid, chid, member\_id}) to check authorization
2. Remove a record with same bid, chid and member\_id from members table
3. Publish event on Kafka
   1. Topic: “channel\_employees\_employee\_unbound”
   2. Content:
      1. user\_id
      2. bid
      3. chid
      4. member\_id
      5. created\_at
4. Return Success

/members/unbindAll

Unbind a list of users from a channel.

### Parameters

* user\_id: String
* bid: String
* chid: String
* member\_ids: String Array

### Returns

* <Basic Response>

### Steps

1. Call Authorization.authorize(“channel\_employees/members/unbind”, {user\_id, bid, chid, member\_ids}) to check authorization
2. For each member\_id in member\_ids list Remove a record with same bid, chid and member\_id from members table
3. Publish event on Kafka
   1. Topic: “channel\_employees\_employee\_group\_unbound”
   2. Content:
      1. user\_id
      2. bid
      3. chid
      4. member\_ids
      5. created\_at
4. Return Success