# Table of Contents

[Table of Contents 1](#_Toc117493040)

[Config File 1](#_Toc117493041)

[Structure of Config File 1](#_Toc117493042)

[.env File 2](#_Toc117493043)

[Structure of .env File 2](#_Toc117493044)

[JSON Requests and Responses 2](#_Toc117493045)

[Requests’s Body that Sent to Server 2](#_Toc117493046)

[Response from Server 2](#_Toc117493047)

[Server API 3](#_Toc117493048)

[Authentification 3](#_Toc117493049)

[Users 3](#_Toc117493050)

[User Groups 6](#_Toc117493051)

[User/Group Permissions 8](#_Toc117493052)

[Info 11](#_Toc117493053)

[Video 13](#_Toc117493054)

[Videostreams 15](#_Toc117493055)

[CRUD-Model of This Application 16](#_Toc117493056)

[Error Codes 16](#_Toc117493057)

# Config File

The file containing the program configuration parameters to be changed is located in the ./configs directory and is called config.yml. If any property from this file is removed or its value is written incorrectly, it will be replaced with a default value on startup with a warning in the console.

## Structure of Config File

db:

connTimeoutSeconds: 5 // how many seconds it takes to try to connect to the database

connLatencyMilliseconds: 50 // how many milliseconds will be the delay in sending each request to the database

host: "127.0.0.1" // what ip is the database server

logConnStatus: false // whether to display the status of connecting / disconnecting from the database in the logs

name: "vhosting" // name of the database in the DBMS

port: "5432" // database server port

sslMode: "disable" // SSL encryption for sending/receiving requests to the database

username: "postgres" // DBMS username

pagination:

getLimitDefault: 100 // default number of items to display in get methods

server:

debugMode: true // set server state to debug mode

maxHeaderBytes: 1048576 // reserve 1 megabyte for storing request headers per server

port: "8000" // web server port

readTimeoutSeconds: 15 // maximum time in seconds to receive responses from the server

writeTimeoutSeconds: 15 // maximum time in seconds to send requests to the server

session:

ttlHours: 168 // browser and session cookie lifetime in hours (which is 7 days)

# .env File

For the server application to work, you need to create an .env file in the /configs directory. By default, it is not bundled with the program for security reasons. It contains information for connecting to your database, a "salt" for hashing passwords, and a signing key for generating and reading session tokens. An example of this arbitrary data file is included in the distribution and is called .env.example.

## Structure of .env File

DB\_DRIVER = "oracle\_db" // name to adjust to your DBMS  
DB\_PASSWORD = "6789" // DBMS user password  
HASHING\_PASSWORD\_SALT = "aB@#$%1" // salt for passwords hashing  
HASHING\_TOKEN\_SIGNING\_KEY = "cD@#$%2" // signing key for generating and reading tokens

# JSON Requests and Responses

## Requests’s Body that Sent to Server

{

"groupIds": [0, 1]

}

*The passed list "[0, 1]" must only be filled with integers >= 0, and must not be empty. (Italics indicate a comment, such as this one.)*

## Response from Server

{

"message": "No groups available."

}

*A message about the successful (non-accidental) completion of the action.*

Or such:

{

"error": {

"errCode": 203,

"statement": "User with entered username is exist."

}

}

*Error message in case of failure of any action with the error code and its interpretation.*

# Server API

## Authentification

POST /auth/signin

POST /auth/change\_password

GET /auth/signout

1. POST /auth/signin

The user logged in.

{

"username": "admin",

"password": "admin"

}

*The "username" and "password" fields are required.*

*==============================================*

{

"message": "You have successfully signed-in."

}

*You are logged in.*

2. POST /auth/change\_password

User changes his password.

{

"username": "admin",

"password": "admin1"

}

*The "username" and "password" fields are required.*

*==============================================*

{

"message": "You have successfully changed password."

}

*You have changed your password.*

3. GET /auth/signout

User logs out.

*Empty request body.*

*==============================*

{

"message": "You have successfully signed-out."

}

*You are logged out.*

## Users

POST /user

GET /user/:id

GET /user/all

POST /user/change\_password

PATCH /user/:id

DELETE /user/:id

1. POST /user

Create user.

{

"username": "admin2",

"password": "admin2",

"firstName": "Ivan",

"lastName": "Ivanov"

}

*The "username" and "password" fields are required.*

*==============================================*

{

"message": "User created."

}

*User has been created.*

2. GET /user/:id

Get user with id from request.

*Empty request body.*

*=========================================================*

{

"id": 1,

"username": "admin2",

"password": "614240232425311c142b2d01aa34e9a36bde480645a57fd69e14155dacfab5a3f9257b77fdc8d8",

"isActive": true,

"isSuperuser": false,

"isStaff": false,

"firstName": "Ivan",

"lastName": "Ivanov",

"joiningDate": "2022-05-19T14:44:15.207391+03:00",

"lastLogin": "2022-05-19T14:44:15.207391+03:00"

}

*The user has been received.*

3. GET /user/all?\_limit=10&\_page=2

Get information about all users in the system.

*You can specify the number of items to display through the parameter after the request: \_limit=<value> and display the same amount (or the remainder) on another page, via the parameter: \_page=<value>.*

*If the page parameter is missing, the “first” page is always displayed, if no parameter is specified, the total number of elements will be displayed (default is 100).*

*Empty request body.*

*=========================================================*

{

"0": {

"id": 0,

"username": "admin",

"password": "614240232425318c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918",

"isActive": true,

"isSuperuser": true,

"isStaff": false,

"firstName": "",

"lastName": "",

"joiningDate": "2022-05-11T09:32:41.115644+03:00",

"lastLogin": "2022-05-19T14:44:11.863332+03:00"

},

"1": {

"id": 1,

"username": "admin2",

"password": "614240232425311c142b2d01aa34e9a36bde480645a57fd69e14155dacfab5a3f9257b77fdc8d8",

"isActive": true,

"isSuperuser": false,

"isStaff": false,

"firstName": "Ivan",

"lastName": "Ivanov",

"joiningDate": "2022-05-19T14:44:15.207391+03:00",

"lastLogin": "2022-05-19T14:44:15.207391+03:00"

}

}

*All users of the system were received.*

4. POST /user/change\_password

Change password for a specific user.

{

"username": "vasya",

"password": "vasya1"

}

*The "username" and "password" fields are required.*

*==============================================*

{

"message": "User password changed."

}

*User password has been changed.*

5. PATCH /user/:id

Partially change user c id from request.

{

"username": "admin2\_edit",

"password": "admin2\_edit",

"isActive": false,

"isSuperuser": true,

"isStaff": false,

"firstName": "Петр",

"lastName": "Петров"

}

*Only those fields are changed, the values of which are not equal to "", or are not specified at all.*

*=====================================================*

{

"message": "User partially updated."

}

*The user has been partially updated.*

6. DELETE /user/:id

Remove user with id from request.

*Empty request body*

*=======================*

{

"message": "User partially updated."

}

*User has been deleted.*

## User Groups

POST /group

GET /group/:id

GET /group/all

PATCH /group/:id

DELETE /group/:id

POST /group/user/:id

GET /group/user/:id

DELETE /group/user/:id

1. POST /group

To create a group.

{

"name": "New Group"

}

*The "name" field is required.*

*==================================*

{

"message": "Group created."

}

*The group was created.*

2. GET /group/:id

Get group with id from request.

*Empty request body.*

*==============*

{

"id": 2,

"name": "New Group"

}

*The group was received.*

3. GET /group/all?\_limit=10&\_page=2

Get information about all groups in the system.

*You can specify the number of items to display through the parameter after the request: \_limit=<value> and display the same amount (or the remainder) on another page, via the parameter: \_page=<value>.*

*If the page parameter is missing, the “first” page is always displayed, if no parameter is specified, the total number of elements will be displayed (default is 100).*

*Empty request body.*

*=================*

{

"0": {

"id": 0,

"name": "Info Writers"

}

}

{

"1": {

"id": 1,

"name": "Video Watchers"

}

}

{

"2": {

"id": 2,

"name": "New Group"

}

}

*All groups received.*

4. PATCH /group/:id

Partially change group c id from request.

{

"name": "Some Group"

}

*Only those fields are changed, the values of which are not equal to "", or are not specified at all.*

*=====================================================*

{

"message": "Group partially updated."

}

*Group partially changed.*

5. DELETE /group/:id

Remove group with id from request.

*Empty request body.*

*==================*

{

"message": "Group deleted."

}

*The group has been deleted.*

6. POST /group/user/:id

Create an association of a user with an id from a request with groups whose id list is passed in the request body.

{

"groupIds": [0, 1]

}

*The passed list "[0, 1]" must only be filled with integers >= 0, and must not be empty.*

*=======================================================*

{

"message": "User groups set."

}

*User has been added to groups.*

7. GET /group/user/:id?\_limit=10&\_page=2

Get a list of id groups to which the user with the id from the request is bound.

*You can specify the number of items to display through the parameter after the request: \_limit=<value> and display the same amount (or the remainder) on another page, via the parameter: \_page=<value>.*

*If the page parameter is missing, the “first” page is always displayed, if no parameter is specified, the total number of elements will be displayed (default is 100).*

*Empty request body.*

*==========================================*

{

"groupIds": [

0,

1

]

}

*The list of id groups in which this user exists was obtained.*

8. DELETE /group/user/:id

Remove the association of a user with an id from a request with a list of group ids passed in the request body.

{

"groupIds": [0, 1]

}

*The passed list "[0, 1]" must only be filled with integers >= 0, and must not be empty.*

*=======================================================*

{

"message": "User groups deleted."

}

*User associations with the listed groups have been removed.*

## User/Group Permissions

GET /perm/all

POST /perm/user/:id

GET /perm/user/:id

DELETE /perm/user/:id

POST /perm/group/:id

GET /perm/group/:id

DELETE /perm/group/:id

1. GET /perm/all?\_limit=10&\_page=2

Get information about all permissions in the system.

*You can specify the number of items to display through the parameter after the request: \_limit=<value> and display the same amount (or the remainder) on another page, via the parameter: \_page=<value>.*

*If the page parameter is missing, the “first” page is always displayed, if no parameter is specified, the total number of elements will be displayed (default is 100).*

*Empty request body.*

*=========================*

{

"0": {

"id": 0,

"name": "Can create a User",

"codeName": "post\_user"

},

"1": {

"id": 1,

"name": "Can get a User",

"codeName": "get\_user"

},

"2": {

"id": 2,

"name": "Can get all of the Users",

"codeName": "get\_all\_users"

},

"3": {

"id": 3,

"name": "Can partially update a User",

"codeName": "patch\_user"

},

"4": {

"id": 4,

"name": "Can delete a User",

"codeName": "delete\_user"

}

}

*All permissions have been obtained for viewing.*

2. POST /perm/user/:id

Create a user id association from a request with a list of permission ids.

{

"permIds": [5, 6, 7]

}

*The passed list "[5, 6, 7]" must only be filled with integers >= 0, and must not be empty.*

====================================================================

{

"message": "User permissions set."

}

*Permissions have been given to the user.*

3. GET /perm/user/:id?\_limit=10&\_page=2

Get list of all permissions associated with user with id from request.

*Empty request body.*

*==========================================================*

{

"permIds": [5, 6, 7]

}

*A list of data for the user with id from the permissions request has been received. If the user does not have personal permissions, "null" is sent instead of the list.*

4. DELETE /perm/user/:id

Remove permission id list associated with user with id from request.

{

"permIds": [5, 6, 7]

}

*The passed list "[5, 6, 7]" must only be filled with integers >= 0, and must not be empty.*

*=========================================================*

{

"message": "User permissions deleted."

}

*User permissions from the list have been removed from the user with id from the request.*

5. POST /perm/group/:id

Create a group association with id from a request with a list of permission ids.

{

"permIds": [4, 2, 1, 3]

}

*The passed list "[4, 2, 1, 3]" should only be filled with integers >= 0, and should not be empty.*

====================================================================

{

"message": "Group permissions set."

}

*Permissions have been given to the group.*

6. GET /perm/group/:id?\_limit=10&\_page=2

Get a list of all permissions associated with a group with id from a request.

*Empty request body.*

*==========================================================*

{

"permIds": [4, 2, 1, 3]

}

*A list of data for the group with id from the permissions request has been received. In case the group does not have permissions, "null" is sent instead of the list.*

7. DELETE /perm/group/:id

Remove permission id list associated with group with id from request.

{

"permIds": [4, 2, 1, 3]

}

*The passed list "[4, 2, 1, 3]" should only be filled with integers >= 0, and should not be empty.*

*=========================================================*

{

"message": "Group permissions deleted."

}

*Group permissions from the list have been removed from the group with id from the request.*

## Info

POST /info

GET /info/:id

GET /info/all

PATCH /info/:id

DELETE /info/:id

1. POST /info

Create info.

{

"stream": "dfmgldmnldndnfgdkfgn",

"startPeriod": "2022-05-18 16:13:51.973640+03",

"stopPeriod": "2022-05-18 16:13:51.973640+03",

"timeLife": "2022-05-18 16:13:51.973640+03"

}

*The "stream" field is required.*

=========================================

{

"message": "Info created."

}

*Info was created.*

2. GET /info/:id

Get info.

*Empty request body.*

*==================================*

{

"id": 1,

"createDate": "2022-05-19T16:37:43.258895+03:00",

"stream": "dfmgldmnldndnfgdkfgn",

"startPeriod": "2022-05-18T16:13:51.97364+03:00",

"stopPeriod": "2022-05-18T16:13:51.97364+03:00",

"timeLife": "2022-05-18T16:13:51.97364+03:00",

"userId": 0

}

*Info has been received.*

3. GET /info/all?\_limit=10&\_page=2

Get all records info in the system.

*You can specify the number of items to display through the parameter after the request: \_limit=<value> and display the same amount (or the remainder) on another page, via the parameter: \_page=<value>.*

*If the page parameter is missing, the “first” page is always displayed, if no parameter is specified, the total number of elements will be displayed (default is 100).*

*Empty request body.*

*===================================*

{

"1": {

"id": 1,

"createDate": "2022-05-19T16:37:43.258895+03:00",

"stream": "dfmgldmnldndnfgdkfgn",

"startPeriod": "2022-05-18T16:13:51.97364+03:00",

"stopPeriod": "2022-05-18T16:13:51.97364+03:00",

"timeLife": "2022-05-18T16:13:51.97364+03:00",

"userId": 0

},

"2": {

"id": 2,

"creationDate": "2022-05-19T16:41:02.697301+03:00",

"stream": "wernwernjwnerjknwerkjnw",

"startPeriod": "2022-05-18T16:13:51.97364+03:00",

"stopPeriod": "2022-05-18T16:13:51.97364+03:00",

"timeLife": "2022-05-18T16:13:51.97364+03:00",

"userId": 0

}

}

*All infohas been received.*

4. PATCH /info/:id

Partially change info.

{

"stream": "qwerty"

"createDate": "2022-04-19T16:41:02.697301+03:00",

"startPeriod": "2022-01-18T16:13:51.97364+03:00",

"stopPeriod": "2022-02-18T16:13:51.97364+03:00",

"timeLife": "2022-03-18T16:13:51.97364+03:00",

"userId": 0

}

*Only those fields are changed, the values of which are not equal to "", or are not specified at all. "userId" does not change.*

======================================================================

{

"message": "Info partially updated."

}

*Info has been partially updated.*

5. DELETE /info/:id

Delete info.

*Empty request body.*

*=================*

{

"message": "Info deleted."

}

*Info has been removed.*

## Video

POST /video

GET /video/:id

GET /video/all

PATCH /video/:id

DELETE /video/:id

1. POST /video

Create video.

{

"url": "https://somewebsite.com/video/knsdkjnfsflsnkn",

"file": "Video1.webm",

"infoId": 1

}

*The fields "url" and "fileName" are required.*

*==========================================*

{

"message": "Video created."

}

*Video was created.*

2. GET /video/:id

Get video.

*Empty request body.*

*==================================*

{

"id": 1,

"url": "https://somewebsite.com/video/knsdkjnfsflsnkn",

"file": "Video1.webm",

"createDate": "2022-05-19T16:49:52.504174+03:00",

"infoId": 1,

"userId": 0

}

*Video has been received.*

3. GET /video/all?\_limit=10&\_page=2

Get all video recordings in the system.

*You can specify the number of items to display through the parameter after the request: \_limit=<value> and display the same amount (or the remainder) on another page, via the parameter: \_page=<value>.*

*If the page parameter is missing, the “first” page is always displayed, if no parameter is specified, the total number of elements will be displayed (default is 100).*

*Empty request body.*

*===================================*

{

"1": {

"id": 1,

"url": "https://somewebsite.com/video/knsdkjnfsflsnkn",

"file": "Video1.webm",

"createDate": "2022-05-19T16:49:52.504174+03:00"

"infoId": 1,

"userId": 0

},

"2": {

"id": 2,

"url": "https://somewebsite.com/video/meqjqnrkjqnr",

"file": "asd.webm",

"createDate": "2022-05-19T16:54:17.666412+03:00"

"infoId": 2,

"userId": 0

}

}

*All videos in the system have been received.*

4. PATCH /video/:id

Partially change the video.

{

"url": "https://somewebsite.com/video/qwerty",

"file": "qwerty.webm",

"createDate": "2022-01-19T16:54:17.666412+03:00"

"infoId": 1,

"userId": 0

}

*Only those fields are changed, the values of which are not equal to "", or are not specified at all. "userId" does not change.*

======================================================================

{

"message": "Video partially updated."

}

*The video has been partially updated.*

5. DELETE /video/:id

Delete video.

*Empty request body.*

=====================

{

"message": "Video deleted."

}

*The video was removed.*

## Videostreams

GET /stream/get/:id

GET /stream/get/all

1. GET /stream/get/:id

Get stream.

*Empty request body.*

*==================================*

{

"id": 1,

"stream": "some stream",

"dateTime": "2022-01-19T16:54:17.666412+03:00",

"statePublic": 1,

"statusPublic": 1,

"statusRecord": 1,

"pathStream": "ksdfkmsfmskfmklsmf"

}

*Video has been received.*

2. GET /stream/get/all?\_limit=10&\_page=2

Get all stream records in the system.

*You can specify the number of items to display through the parameter after the request: \_limit=<value> and display the same amount (or the remainder) on another page, via the parameter: \_page=<value>.*

*If the page parameter is missing, the “first” page is always displayed, if no parameter is specified, the total number of elements will be displayed (default is 100).*

*Empty request body.*

*===================================*

{

"1": {

"id": 1,

"stream": "some stream 1",

"dateTime": "2022-01-19T16:54:17.666412+03:00",

"statePublic": 1,

"statusPublic": 1,

"statusRecord": 1,

"pathStream": "ksdfkmsfmskfmklsmf"

},

"2": {

"id": 2,

"stream": "some stream 2",

"dateTime": "2022-01-20T16:54:17.666412+03:00",

"statePublic": 1,

"statusPublic": 1,

"statusRecord": 1,

"pathStream": "asdadandkjnqkwnekqne"

}

}

*All streams in the system have been received.*

# CRUD-Model of This Application

POST /auth/signin --> Auth.SignIn()

POST /auth/change\_password --> Auth.ChangePassword()

POST /auth/signout --> Auth.SignOut()

POST /user --> User.CreateUser()

GET /user/:id --> User.GetUser()

GET /user/all --> User.GetAllUsers()

POST /user/change\_password --> User.UpdateUserPassword()

PATCH /user/:id --> User.PartiallyUpdateUser()

DELETE /user/:id --> User.DeleteUser()

POST /group --> Group.CreateGroup()

GET /group/:id --> Group.GetGroup()

GET /group/all --> Group.GetAllGroups()

PATCH /group/:id --> Group.PartiallyUpdateGroup()

DELETE /group/:id --> Group.DeleteGroup()

POST /group/user/:id --> Group.SetUserGroups()

GET /group/user/:id --> Group.GetUserGroups()

DELETE /group/user/:id --> Group.DeleteUserGroups()

GET /perm/all --> Perm.GetAllPermissions()

POST /perm/user/:id --> Perm.SetUserPermissions()

GET /perm/user/:id --> Perm.GetUserPermissions()

DELETE /perm/user/:id --> Perm.DeleteUserPermissions()

POST /perm/group/:id --> Perm.SetGroupPermissions()

GET /perm/group/:id --> Perm.GetGroupPermissions()

DELETE /perm/group/:id --> Perm.DeleteGroupPermissions()

POST /info --> Info.CreateInfo()

GET /info/:id --> Info.GetInfo()

GET /info/all --> Info.GetAllInfos()

PATCH /info/:id --> Info.PartiallyUpdateInfo()

DELETE /info/:id --> Info.DeleteInfo()

POST /video --> Video.CreateVideo()

GET /video/:id --> Video.GetVideo()

GET /video/all --> Video.GetAllVideos()

PATCH /video/:id --> Video.PartiallyUpdateVideo()

DELETE /video/:id --> Video.DeleteVideo()

GET /stream/get/:id --> Stream.GetStream()

GET /stream/get/all --> Stream.GetAllStreams()

# Error Codes

100. User with entered username or password is not exist.

A database search was performed for the username entered and the hashed password, but returned no results.

101. Cannot generate token. Error: <debug\_error>.

Unable to generate token. This is purely a logical error on the server side. Most likely this will never happen.

102. You must be signed-in for changing password.

You have not been authorized to change your password (your token does not have a copy in the form of a session in the database, or you simply do not have one).

103. Cannot parse token. Error: <debug\_error>.

It is impossible to decrypt the token due to an error in its encryption.

104. User with entered username is not exist.

The user with the entered username is not in the database.

105. Password cannot be empty.

The password you entered is an empty string.

106. Cannot update user password. Error: <debug\_error>.

A database error will be thrown if the user's password update fails.

107. You must be signed-in for signing-out.

You simply cannot log out of the system until you are logged in.

108. User with this username doesn't exist.

The user on whose behalf you are going to act does not exist in the system.

200. Cannot bind input data. Error: <debug\_error>.

Error in the content of the sent data in JSON format in the body of the request to the server. Check if all the entries you have specified match their data types.

201. Username and password cannot be empty.

You left username or password blank so you can't continue.

202. Cannot check user existence. Error: <debug\_error>.

A database error will be thrown if the user existence check fails. Under the hood, it can check for the existence of a record in the database by user id or username.

203. User with entered username is exist.

You cannot create a user with the entered username because it is already in use by another user.

204. Cannot create user. Error: <debug\_error>.

The DB error will be thrown when the user is created.

205. Cannot convert requested ID to type int. Error: <debug\_error>.

If you enter something other than an integer as an identifier (id) in the request URL, you will get this error.

206. User with requested ID is not exist.

The user by the requested ID is not in the database, which means it is not in the system either.

207. Cannot get user. Error: <debug\_error>.

A DB error will be thrown when getting the user.

208. Cannot get all users. Error: <debug\_error>.

DB error will be thrown when getting all users.

209. Cannot partially update user. Error: <debug\_error>.

A database error will be thrown when the user is partially modified.

210. Cannot delete user. Error: <debug\_error>.

A database error will be thrown when the user is deleted.

211. You don't have enough permissions.

The server checked your cookies, if there was a token, it decrypted it and searched for the session; if there is a session, then it was checked who you are by the isSuperuser or isStaff flags, and if you did not have one of these 2 flags, the server looked among your personal permissions for one that would allow you to continue. But, as you can see from the error, I did not find it.

212. Cannot check superuser/staff permissions. Error: <debug\_error>.

A database error will be thrown when looking for the isSuperuser or isStaff flag.

213. Cannot check personal permission. Error: <debug\_error>.

A database error will be thrown when looking for your personal permission for a specific action on the server.

214. User with entered username is not exist.

You have entered a username that does not exist, so you cannot continue.

300. Cannot delete session. Error: <debug\_error>.

A database error will be thrown when you try to delete your current session.

301. Cannot create session. Error: <debug\_error>.

A database error will be thrown when trying to create a new session.

302. Cannot get session and date. Error: <debug\_error>.

A database error will be thrown when looking up your last session and the date it was created.

400. Cannot do logging. Error: <debug\_error>.

Such an error cannot be reflected in the logs in the database. It is present only in the console when the programmer forgot to write a constant with the type of a new structure to compare the output of a newly created module in the system.

500. Group name cannot be empty.

When creating a group, you must not leave the name field empty.

501. Cannot check group existence. Error: <debug\_error>.

A database error will be thrown when checking for the group's existence.

502. Group with entered name is exist.

You cannot create a group whose name is the same as another already created group.

503. Cannot create group. Error: <debug\_error>.

A database error will be thrown when the group is created.

504. Group with requested ID is not exist.

The group you tried to get via id in the request is not in the system.

505. Cannot get group. Error: <debug\_error>.

A database error will be thrown when getting the group.

506. Cannot get all groups. Error: <debug\_error>.

A database error will be thrown when all groups are retrieved.

507. Cannot partially update group. Error: <debug\_error>.

The database error will be displayed when the group is partially updated.

508. Cannot delete group. Error: <debug\_error>.

A database error will be thrown when deleting a group.

509. Group IDs cannot be empty.

The passed id list cannot be empty.

510. Cannot set user groups. Error: <debug\_error>.

A database error will be thrown when adding a user to groups.

511. Cannot get user groups. Error: <debug\_error>.

A database error will be thrown when taking a list of groups the user is a member of.

512. Cannot delete user groups. Error: <debug\_error>.

A database error will be displayed when a user is removed from the groups specified in the list.

600. Cannot get all permissions. Error: <debug\_error>.

A database error will be thrown when getting a list of all possible action permissions.

601. Permission IDs cannot be empty.

The passed list of permissions cannot be empty.

602. Cannot set user permissions. Error: <debug\_error>.

A DB error will be thrown when setting the permission list for a user.

603. Cannot get user permissions. Error: <debug\_error>.

A database error will be thrown when getting a list of the user's personal permissions.

604. Cannot delete user permissions. Error: <debug\_error>.

A database error will be thrown when the user's permission list is deleted.

605. Cannot set group permissions. Error: <debug\_error>.

A database error will be thrown when setting the permission list for groups.

606. Cannot get group permissions. Error: <debug\_error>.

A database error will be thrown when getting a list of permissions for a particular group.

607. Cannot delete group permissions. Error: <debug\_error>.

A database error will be thrown when the group's permission list is settled.

700. Stream cannot be empty.

The stream field must not be empty.

701. Cannot create info. Error: <debug\_error>.

DB error will be thrown when creating info.

702. Cannot check info existence. Error: <debug\_error>.

A database error will be thrown when checking for the existence of info.

703. Info with requested ID is not exist.

Info with the requested id does not exist in the system.

704. Cannot get info. Error: <debug\_error>.

DB error will be thrown when info is received.

705. Cannot get all infos. Error: <debug\_error>.

A database error will be thrown when all info is received.

706. Cannot partially update info. Error: <debug\_error>.

The database error will be displayed when the info is partially changed.

707. Cannot delete info. Error: <debug\_error>.

DB error will be thrown when deleting info.

800. URL and file name cannot be empty.

The url and fileName fields must be filled in.

801. Cannot create video. Error: <debug\_error>.

DB error will be thrown when creating video

802. Cannot check video existence. Error: <debug\_error>.

A database error will be thrown when checking if the video exists.

803. Video with requested ID is not present.

The video with the requested id does not exist in the system.

804. Cannot get video. Error: <debug\_error>.

DB error will be thrown when receiving video.

805. Cannot get all videos. Error: <debug\_error>.

DB error will be thrown when all videos are retrieved.

806. Cannot partially update video. Error: <debug\_error>.

The database error will be displayed when the video is partially updated.

807. Cannot delete video. Error: <debug\_error>.

DB error will be thrown when deleting video.