Federnet Specification

Version 1.7

The following specification defines the API behind the Federnet network, the types of participants on the network, and the times when different requests should be sent. It also provides implementation requirements for some requests. Although these requirements might not directly relate to the API itself, they must be adhered to for a valid implementation of this specification.

Objects

The API uses the following objects which will be sent in the request / response body:

Account

Represents a user account.

```
{
  "username": string,
  "password": string
}
```

Community

Represents a community on the network, in which users can make posts.

```
{
  "name": string,
  "description": string,
  "address": string,
  "publicKey": string
}
```

`address` contains the web address of the server hosting the community, this can be an IP address or DNS address. The address can optionally specify a port using the standard <address>:<port> syntax. If the port is not provided, a default port will be assumed.

Post

Represents a post made by a user in a specific community.

```
{
  "posterUsername": string,
  "content": string,
  "timestamp": integer
}
```

`timestamp` contains a UNIX timestamp representing the time that the post was received by the server.

DirectMessage

Represents a private message sent by one user to another.

```
{
  "senderUsername": string,
  "recipientUsername": string,
  "content": string,
  "timestamp": integer
}
```

timestamp contains a UNIX timestamp representing the time that the message was received by the server.

Participants and Devices

The following types of devices will participate on the network:

Infrastructure Server

These are operated by Federnet itself, and provide user account and community discovery functionality.

Community Server

These are owned and operated by community operators, and are outside of the Federnet operator’s control. Each community must provide a community server on which it can be hosted. All posts made in a community are hosted on the community server belonging to that community.

User Device

These are the devices used by the users to connect to Federnet

Response Format

The response bodies will contain information in the following format:

```
{
  "code": integer,
  "message": string,
  "data": []
}
```

message contains a human readable message, either explaining that the request was successful or why it failed.

data contains a list containing response data specific to the type of request. For requests that don’t need response data, it is omitted.

code contains a number specifying whether the request was successful or why it was not. **This is not the HTTP status code, but a Federnet specific code.** The purpose of this number is to allow implementations a more reliable way to decide what to do based on the value of this field compared with checking the message field, which is intended for humans. This allows more detailed information to be provided in message without making the internal comparison logic more difficult.

Codes

The list of all codes is defined here:

Code	Name	Description	Request types
0	Success	The request succeeded	All
1	GenericFailure	A generic catch all code	All
2	UsernameNotUnique	Could not create account as the provided username is not unique.	CreateAccount
3	UnsuitablePassword	Could not create account as the provided password does not meet complexity requirements.	CreateAccount
4	BadCredentials	Username or password is incorrect.	GetSession
5	UserNotFound	No user exists with the specified name	GetPublicKey, SendDirectMessage
6	NoPublicKey	The specified user does not have a public key.	GetPublicKey, SendDirectMessage
7	CommunityNameNotUnique	Could not register community as the provided name is not unique.	RegisterCommunity, UpdateCommunity

8	UnsuitableAddress	The server at the provided address either couldn't be reached, or did not identify itself as a community server.	RegisterCommunity, UpdateCommunity
9	UnauthorisedCommunityRequest	Verification of the request's signature failed.	UpdateCommunity, RemoveCommunity
10	CommunityNotFound	Infrastructure server has no record of a community with the specified name.	UpdateCommunity, RemoveCommunity, GetCommunityInfo
11	StaleRequest	The timestamp of the request is too old.	UpdateCommunity, RemoveCommunity
12	UnauthorisedUserRequest	Verification of the user's JWT token failed	SetPublicKey, CreatePost, FetchPosts, SendDirectMessage, FetchDirectMessages

For any requests which have required fields missing, or invalid or misformed data, a `400 - Bad Request` status code is returned and a `GenericFailure` code is used in the body.

Requests

All requests will be sent over HTTPS, and will therefore be encrypted.

Accounts

CreateAccount

Implementation Requirement: The password itself must not be stored by the server; a salted hash must be used instead.

Request

Description	Requests that a new user account be created by the Infrastructure Server
Sender	User Device
Recipient	Infrastructure Server
Sent when	The user wants to create a new account
Endpoint	`/accounts`
Request Method	POST

Headers

No additional headers needed

Body

A single **Account** object, with all fields filled in.

Response

If the username is not unique:

Response code: `409 - Conflict`

Reponse body:

```
{
  "code": <UsernameNotUnique>,
  "message": "Username is not unique"
}
```

If the password does not meet complexity requirements:

The password requirements are to be decided by the implementation.

Response code: `403 - Forbidden`

Response body:

```
{
  "code": <UnsuitablePassword>,
  "message": "Password does not meet complexity requirements"
}
```

`message` should also contain the complexity requirements.

Successfully created account:

Response code: `200 - Ok`

Response Body:

```
{
  "code": <Success>,
  "message": "Account created sucessfully"
}
```

GetSession

Request

Description	Used to obtain an authentication token to verify that the user is logged in
Sender	User Device
Recipient	Infrastructure Server
Sent when	The user wants to log in to the network. A user’s existing token is about to expire and a new one is needed.
Endpoint	`/sessions`
Request Method	POST

Headers

No additional headers needed

Body

A single **Account** object, with `username` and `password` fields filled in

Response

If username or password don’t match an account on the server:

Response code: `401 - Unauthorized`

Response body:

```
{
  "code": <BadCredentials>,
  "message": "Incorrect username or password"
}
```

If username and password are correct:

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Log in successful",
  "data": [
    <JWT Token>
  ]
}
```

JWT token fields:

```
{
  "username": string,
  "exp": integer,
  "publicKeyHash": string
}
```

`exp` contains the time that the token should expire, represented as a unix timestamp.
`publicKeyHash` contains the MD5 hash digest of the Infrastructure Server's public key represented as a base64 string.

SetPublicKey

Request

Description	Sets a new public key for the user, for use in Direct Messaging
Sender	User Device
Recipient	Infrastructure Server
Sent when	The user wants to use direct messaging functionality, so is setting up a public key. The user has a new public key.
Endpoint	`/accounts/<username>`
Request Method	PUT

Headers

Header	Value	Description
Authorization	Bearer <User's JWT>	Allows the infrastructure server to verify that the user is logged in.

Body

```
{
  "publicKey": string
}
```

`public_key` contains the public key from an RSA keypair. The corresponding private key should be kept safe and secure by the user client as it will be needed to decrypt messages.

Response

The username from the URL should be ignored, instead the username from the JWT should be used. It is only provided in the URL for a more RESTful URL pattern.

If JWT validation fails:

Response code: `401 - Unauthorised`

Response body:

```
{
  "code": <UnauthorisedUserRequest>
  "message": "User is not authorised"
}
```

If key is set successfully:

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Sucessfully set public key"
}
```

GetPublicKey

Request

Description	Gets a user’s public key
Sender	User Device
Recipient	Infrastructure Server
Sent when	Each time the user wants to send a Direct Message to another user this is used to get the recipient’s public key
Endpoint	`/accounts/<username>`
Request Method	GET

Headers

No additional headers needed

Response

<u>If username does not exist:</u>

Response code: `404 - Not Found`

Response body:

```
{
  "code": <UserNotFound>,
  "message": "No user found with the specified name"
}
```

<u>If user does not have a public key:</u>

Response code: `404 - Not Found`

Response body:

```
{
  "code": <NoPublicKey>,
  "message": "The specified user has not setup a public key"
}
```

<u>If public key fetched successfully:</u>

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>
  "message": "Fetched public key",
  data: [
    <User's public key>
  ]
}
```

Communities

Ping

Description	Checks if there is a community server at an address
Sender	User Device, Infrastructure server
Recipient	Community Server
Sent when	The infrastructure server or a user device wants to check if an address points to a community server.
Endpoint	`/ping`
Request Method	GET

Response

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Pong"
}
```

RegisterCommunity

Request

Description	Joins a new community to the network
Sender	Community Server
Recipient	Infrastructure Server
Sent when	A new community is being created
Endpoint	`/communities`
Request Method	POST

Headers

No additional headers needed

Body

A single **Community** object with all the fields filled in.

Before sending the request, an RSA keypair should be generated. The private key should be permanently stored somewhere secure (as it will be needed for any further requests relating to the administration of the community server). The public key should be provided in the `publicKey` field of the request.

Response

If the name is not unique:

Response code: `409 - Conflict`

Response body:

```
{
  "code": <CommunityNameNotUnique>,
  "message": "Community name is not unique"
}
```

If the address isn't a community server:

The infrastructure server should send a ping request to the specified address. This response is used if it fails to receive a valid response back.

Response code: `403 - Forbidden`

Response body:

```
{
  "code": <UnsuitableAddress>
  "message": "The server at the given address failed to identify itself
as a Federnet community server"
}
```

If community is registered successfully:

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>
  "message": "Successfully registered community"
}
```

UpdateCommunity

Request

Description	Updates the info the Infrastructure Server has on a community
Sender	Community Server
Recipient	Infrastructure Server
Sent when	Community name changes. Community description changes. A Community Server moves to a new address
Endpoint	`/communities/<community name>`
Request Method	PATCH

Headers

No additional headers needed

Body

A single **Community** object, containing only the fields to be changed, as well as the following fields:

```
{
  "timestamp": integer,
  "signature": string
}
```

`timestamp` contains a UNIX timestamp representing the time that the request was created.

`signature` contains an RSA cryptographic signature of the rest of the body

To obtain the value for `signature`:

1. Combine the fields of the request body (including the timestamp) into a single string as follows:

- Order the fields in ascending order by the sum of the Unicode character codes of each character in the field name

- Combine the fields into a string in the format: "<field1 name>:<field1 value>,<field2 name>:<field2 value>,…,<field n name>:<field n value>"
- e.g. `description:A new description,timestamp:1674398640`

2. Take the SHA256 hash of the combined string

3. Use the community's private key to encrypt the hash, and encode the ciphertext in base64

Response

Verifying the `signature`

Upon receiving the request, the infrastructure server should take a SHA256 hash of the request body fields (not including the `signature` field) in the string format specified above. It should then use the community's public key to decrypt the signature. The decrypted hash should be identical to the hash that the server just calculated (meaning that the request has not been modified, and was signed with the correct private key).

If the specified community does not exist:

Response code: `404 - Not Found`

Response body:

```
{
  "code": <CommunityNotFound>,
  "message": "No community found with the specified name"
}
```

If the hashes don't match:

Response code: `401 - Unauthorized`

Response body:

```
{
  "code": <UnauthorisedCommunityRequest>,
  "message": "Failed to verify signature"
}
```

If `timestamp` is more than 60 seconds in the past:

To avoid old UpdateRequests being intercepted and stored and then later resent by a malicious party, requests with a `timestamp` value more than 60 seconds old will not be acted upon.

Response code: `403 - Forbidden`

Response body:

```
{
  "code": <StaleRequest>,
  "message": "Request timestamp is too old"
}
```

If a name has been provided and is not unique:

Response code: `409 - Conflict`

Response body:

```
{
  "code": <CommunityNameNotUnique>,
  "message": "Community name is not unique"
}
```

<u>If an address has been provided but isn’t a community server:</u>

The infrastructure server should send a ping request to the specified address. This response is used if it fails to receive a valid response back.

Response code: `403 - Forbidden`

Response body:

```
{
  "code": <UnsuitableAddress>
  "message": "The server at the given address failed to identify itself
as a Federnet community server"
}
```

<u>If the community is updated successfully:</u>

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Successfully updated community info"
}
```

RemoveCommunity

Request

Description	Removes a community from the Infrastructure server’s list of communities
Sender	Community Server
Recipient	Infrastructure Server
Sent when	The community is closing down
Endpoint	`/communities/<community name>`
Request Method	DELETE

<u>Headers</u>

No additional headers needed

<u>Body:</u>

```
{
  "timestamp": integer,
  "signature": string
}
```

The `signature` should be calculated in the same way as in **UpdateCommunity**

Response

The signature should be verified in the same way as in **UpdateCommunity**

If the specified community does not exist:

Response code: `404 - Not Found`

Response body:

```
{
  "code": <CommunityNotFound>,
  "message": "No community found with the specified name"
}
```

If the hashes don’t match:

Response code: `401 - Unauthorized`

Response body:

```
{
  "code": <UnauthorisedCommunityRequest>,
  "message": "Failed to verify signature"
}
```

If `timestamp` is more than 60 seconds in the past:

To avoid old UpdateRequests being intercepted and stored and then later resent by a malicious party, requests with a `timestamp` value more than 60 seconds old will not be acted upon.

Response code: `403 - Forbidden`

Response body:

```
{
  "code": <StaleRequest>,
  "message": "Request timestamp is too old"
}
```

If the community is removed successfully:

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Successfully removed community"
}
```

FetchCommunities

Request

Description	Fetches the list of communities
Sender	User Device
Recipient	Infrastructure Server
Sent when	The user wants to discover new communities
Endpoint	`/communities`
Request Method	GET

Headers

No additional headers needed

Response

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Fetched communities list",
  "data": [
    <List of Community objects>
  ]
}
```

GetCommunityInfo

Request

Description	Gets info about a specific community
Sender	User Device
Recipient	Infrastructure server
Sent when	The user device wants to check that it has the correct address for a community
Endpoint	`/communities/<community name>`
Request Method	GET

Headers

No additional headers needed

Response

If specified community does not exist:

Response code: `404 - Not found`

Response body:

```
{
  "code": <CommunityNotFound>,
  "message": "No community found with the specified name"
}
```

If community exists:

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Got community info",
  "data": [
    <A single community object containing the community info>
  ]
}
```

Posts

When handling requests that require Authorization headers, the Community Server must verify that the sender is who they claim to be by verifying the JWT provided with the request. The server must verify the JWT using the Infrastructure Server's public key.

To ensure that the Community Server is aware if the Infrastructure Server's public key changes, the Community Server must compare the `publicKeyHash` field of the JWT to an MD5 hash digest of the value that it currently has for the Infrastructure Server's public key (this comparison only needs to be done if a JWT fails verification). If the digests do not match then it is possible (but not certain, as there is no guarantee that the JWT's `publicKeyHash` field is legitimate) that the Community Server has an outdated key, so the Community Server must send a **GetInfrastructureServerPublicKey** request to find out if the key has changed.

CreatePost

Request

Description	Submits a new post to a community
Sender	User Device
Recipient	Community Server
Sent when	The user wants to make a new post in a community
Endpoint	`/posts`
Request Method	POST

Headers

Header	Value	Description
Authorization	Bearer <User’s JWT>	Allows the community server to verify that the user is logged in.

Body

A single **Post** object, without the `timestamp` or `username` fields (as the poster username can be obtained from the JWT and the timestamp should be added by the server)

Response

If JWT validation fails:

Response code: `401 - Unauthorized`

Response body:

```
{
  "code": <UnauthorisedUserRequest>
  "message": "User is not authorised"
}
```

If post is submitted successfully:

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Successfully submitted post"
}
```

FetchPosts

Request

Description	Fetches posts from a community that were made between two specified times
Sender	User Device
Recipient	Community Server
Sent when	The user wants to see the latest posts from a community. The user is looking back on less recent posts from a community.
Endpoint	`/posts`
Request Method	GET

Headers

Header	Value	Description
Authorization	Bearer <User’s JWT>	Allows the community server to verify that the user is logged in.
Start-Time	<UNIX Timestamp>	Posts from before this timestamp won’t be returned

End-Time	<UNIX timestamp>	Posts from after this timestamp won't be returned. (Optional. If not provided, current timestamp will be used)

Response

<u>If JWT validation fails:</u>

Response code: `401 - Unauthorized`

Response body:

```
{
  "code": <UnauthorisedUserRequest>
  "message": "User is not authorised"
}
```

<u>If posts fetched succesfully:</u>

The implementation may choose to limit the number of posts that are returned per fetch request in order to avoid overloading resources. This limit is for the implementation to decide, however if doing so it must return the last N posts before (or equal to) the End-Time (as opposed to the first N posts after the Start-Time). The posts should still be returned in ascending chronological order.

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Fetched posts",
  "data": [
    <List of Post objects>
  ]
}
```

Direct Messages

SendDirectMessage

Request

Description	Sends a direct message for another user to the server
Sender	User Device
Recipient	Infrastructure Server
Sent when	The user wants to send a direct message to another user.
Endpoint	`/DirectMessages`
Request Method	POST

<u>Headers</u>

Header	Value	Description

Authorization	Bearer <User’s JWT>	Allows the infrastructure server to verify that the user is logged in.

Body

Before sending, the client should request the recipient’s public key from the infrastructure server using **GetPublicKey**. This key should be used to encrypt the `content` field. Once encrypted, the content should be represented as a base64 encoded string.

The body should contain a **DirectMessage** object, with all fields but the `timestamp` and `senderUsername` filled in. The `timestamp` will be added by the infrastructure server upon receipt, and the server will add the `senderUsername` from the username in the JWT.

Response

If JWT validation fails:

Response code: `401 - Unauthorized`

Response body:

```
{
  "code": <UnauthorisedUserRequest>,
  "message": "User is not authorised"
}
```

If the `recipientUsername` does not match an actual user:

Response code: `403 - Forbidden`

Response body:

```
{
  "code": <UserNotFound>,
  "message": "No recipient found with the specified name"
}
```

If the recipient does not have a public key:

Response code: `403 - Forbidden`

Response body:

```
{
  "code": <NoPublicKey>,
  "message": "Recipient has not set a public key so cannot receive
direct messages"
}
```

If the message is stored successfully:

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Successfully sent direct message"
}
```

FetchDirectMessages

Request

Description	Fetches all direct messages for which the user is the recipient between two times
Sender	User Device
Recipient	Infrastructure Server
Sent when	The user wants to see what direct messages they have received.
Endpoint	`/DirectMessages`
Request Method	GET

Headers

Header	Value	Description
Authorization	Bearer <User's JWT>	Allows the infrastructure server to verify that the user is logged in.
Start-Time	<UNIX timestamp>	Messages from after this timestamp won't be returned.
End-Time	<UNIX timestamp>	Messages from after this timestamp won't be returned. (Optional. If not provided, current timestamp will be used)

Response

If JWT validation fails:

Response code: `401 - Unauthorized`

Response body:

```
{
  "code": <UnauthorisedUserRequest>,
  "message": "User is not authorised"
}
```

If messages are fetched succesfully:

The implementation may choose to limit the number of messages that are returned per fetch request in order to avoid overloading resources. This limit is for the implementation to decide, however if doing so it must return the last N messages before (or equal to) the End-Time (as opposed to the first N messages after the Start-Time). The messages should still be returned in ascending chronological order.

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Successfully fetched direct messages",
  "data": [
    <List of DirectMessage objects>
  ]
}
```

Other

GetInfrastructureServerPublicKey

Request

Description	Get the Infrastructure Server’s public key
Sender	Community Server
Recipient	Infrastructure Server
Sent when	A Community Server needs to obtain the Infrastructure Server’s public key for use in JWT verification.
Endpoint	`/publicKey`
Request Method	GET

Headers

No additional headers needed

Response

Response code: `200 - Ok`

Response body:

```
{
  "code": <Success>,
  "message": "Successfully fetched public key",
  "data": [
    <Infrastructure Server's public key>
  ]
}
```