Server API is based on HTTPS and JSON. Clients send messages to the server following the API, and handle responses from the server.

For overview and further details on the requirements of the system, see the course project work requirements elsewhere. This document only documents the API that must be supported by server and client implementations.

Please note that automatic tests will be conducted to verify the functionality. Therefore, both the server and the client MUST follow this API. Failing to do so fails the tests. If specific input/output is given in the requirements, it must be the same in your coursework.

Assignment specifications

Minimum requirements

This section provides the minimum requirements for the coursework, ALL requirements must be completed

REQ1 All text in HTTP body MUST be UTF-8.

REQ2 All content in the HTTP requests and responses MUST be JSON except for the error messages from server which may be text.

REQ3 All dates and times in JSON content MUST follow the ISO 8601 date format in UTC time zone with milliseconds, e.g. 2020-12-21T07:57:47.131Z.

REQ4 All times must follow the following pattern: "2020-12-21T07:57:47.123Z"

REQ5 HTTP Headers MUST contain the content size in bytes and content type using standard HTTP headers.

REQ6 HTTP header Content-Type MUST be "application/json".

REQ7 Database MUST be used to store all user sent data

1.1. Feature 1: User can store observation information as a record

URL: https://server.url/datarecord

Request type: POST

Response code for successfully operation: 200

Response code failed operation: 400-499

Header information that must be included in minimum:

Content-Length: size of the content

Content-Type: application/json

Authorization: only registered users can perform this operation

Data format / payload for posting a sightseeing location:

{

“recordIdentifier” : “Jupiter”,

“recordDescription”: “Notes about Jupiter during observation”,

“recordPayload”: “Jupiter great red spot noticed, no changes related to previous observation”,

“recordRightAscension” : “05h 11m 42s”,

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Matti”

}

Response payload:

- N/A

Where:

- recordIdentifier is an unique identifier given by the user
- recordDescription short title given by user about the record
- recordPayload any information given by the user about the record (text, numbers, binary representation of picture….)

- recordRightAscension Right ascension (abbreviated RA) is similar to longitude and is measured in hours, minutes and seconds eastward along the celestial equator. The distance around the celestial equator is equal to 24 hours.
- recordDeclination Declination is similar to latitude and is measured in degrees, arcminutes and arcseconds, north or south of the celestial equator. Positive values for declination correspond to positions north of the equator, while negative values refer to positions south of the equator. The declination of the north celestial pole is 90° 0' 0" and the south celestial pole's declination is -90° 0' 0". The equator is 0° 0' 0".
- recordOwner is the name of the person who sent the location information (user's chosen nickname taken from the user's profile, not to be mixed with login username)

NOTE: If the server receives messages with same identifier, it needs to be stored as well.

1.2. Feature 2: Get all observations from the server

URL: https://server.url/datarecord

Request type: GET

Response code for successfully operation: 200

Response code failed operation: 400-499

Header information that must be included in minimum:

- Content-Length: size of the content
- Content-Type: application/json

Authorization: only registered users can perform this operation

Response payload:

Example data format for sending locations back to client:

[

{

“recordIdentifier” : “Jupiter”,

“recordDescription”: “Notes about Jupiter during observation”,

“recordPayload”: “Jupiter great red spot noticed, no changes related to previous observation”,

“recordRightAscension” : “05h 11m 42s”,

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Matti”,

“recordTimeRecieved” : “2024-12-21T07:57:47.123Z”

},

{

“recordIdentifier” : “Mars”,

“recordDescription”: “Notes about Mars during observation”,

“recordPayload”: “Day 1345: Still no sighting of Elon Musk’s colony”,

“recordRightAscension” : “05h 11m 42s”,

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Pekka”,

“recordTimeRecieved” : “2024-12-21T07:57:47.123Z”

}

]

Where,

- recordTimeRecieved is a server provided timestamp when a message was originally received

NOTE: Server MAY also response with 204 and empty response body if there are no coordinates to deliver to the client.

1.3. Feature 3: Register and authenticate

URL: https://server.url/registration

Request type: POST

Response code for successfully operation: 200

Response code failed operation: 400-499

Header information that must be included in minimum:

- Content-Length: size of the content
- Content-Type: application/json

Authorization: N/A (user can register an account without previous credentials)

Data format for registering user:

{

“username” : “username”,

“password” : “password”,

“email” : “user.email@for-contacting.com”,

“userNickname” : “Pekka”

}

Where

- username is the account name of the user, it has to be unique
- password is user chosen password for the account
- email is email address of user, must be unique and must be in following pattern:
    - string@string.string
- userNickname is a default user nickname user wants to use when sending observations, not unique and can change

Description:

Server MUST NOT accept other than registration requests from unauthorized clients. Client MUST send passwords using HTTPS. Passwords MUST not be sent over insecure HTTP connections. Server MUST not store the plain password, but only store a hash of it. A good hash function MUST be used to make sure hashed passwords are secured and as unique as possible to avoid collisions.

For other than registration requests, users MUST be authenticated using Basic HTTP authentication (see References). Authentication MUST be done over HTTPS using the Authorization header, where username:password string MUST be UTF-8.

1.4. Feature 4: User can attach identifying observation information to the observation data

URL: https://server.url/datarecord

Request type: POST

Response code for successfully operation: 200

Response code failed operation: 400-499

Header information that must be included in minimum:

- Content-Length: size of the content
- Content-Type: application/json

Authorization: only registered users can perform this operation

Data format for posting a sightseeing location:

{

“recordIdentifier” : “Jupiter”,

“recordDescription”: “Notes about Jupiter during observation”,

“recordPayload”: “Jupiter great red spot noticed, no changes related to previous observation”,

“recordRightAscension” : “05h 11m 42s”,

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Matti”,

“observatory” :

[

{

“observatoryName”:”Puolivälinkankaan vesitornin observatorio”,

“latitude”: “65.0580507136066”,

“longitude”: “25.469414591526057”,

}

]

}

Where:

- latitude and longitude represent a geolocation based on java html geolocation api (such as used in google maps for example), expected datatype is Double

NOTE: Observatory information is not mandatory, if no observatory exist for a sighting message, no observaotry information is provided in a message when requested from the server.

Example return payload:

Example data format for sending locations back to client:


```
[
        {
                **“recordIdentifier” : “Jupiter”,**
                **“recordDescription”: “Notes about Jupiter during observation”,**
                **“recordPayload”: “Jupiter great red spot noticed, no changes related to previous observation”,**
                **“recordRightAscension” : “05h 11m 42s”,**
                **“recordDeclination”: “+22° 15' 54””,**
                **“recordOwner” : ”Matti”,**
                **“recordTimeRecieved” : “2024-12-21T07:57:47.123Z”,**

                “observatory” :
                [
                        {
                        “observatoryName”:”Puolivälinkankaan vesitornin observatorio”,
                        “latitude”: “65.0580507136066”,
                        “longitude”: “25.469414591526057”,
                        }
                ]

        }

]
```

Advanced API

Any of the following features successfully implemented (passes unit tests and matches requirements) increases the assignment grade by 1. For example, if two additional features are implemented successfully, the grade of the assignment 3. If all exercises are also successfully completed, the final course grade is 4.

Feature 5: Contact FMI mock service to get observation location weather information

URL: https://server.url/info

Request type: POST

Response code for successfully operation: 200

Response code failed operation: 400-499

Header information that must be included in minimum:

- Content-Length: size of the content
- Content-Type: application/json

Authorization: only registered users can perform this operation

Data format for posting a sightseeing location:

{

“recordIdentifier” : “Jupiter”,

“recordDescription”: “Notes about Jupiter during observation”,

“recordPayload”: “Jupiter great red spot noticed, no changes related to previous observation”,

“recordRightAscension” : “05h 11m 42s”,

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Matti”,

“recordTimeRecieved” : “2024-12-21T07:57:47.123Z”,

“observatory” :

[

{

“observatoryName”:”Puolivälinkankaan vesitornin observatorio”,

“latitude”: “65.0580507136066”,

“longitude”: “25.469414591526057”,

}

],

“observatoryWeather”:[]

}

Where weather is:

{

“temperatureInKelvins”:” 253.15”,

“cloudinessPercentance”: “0”,

“bagroundLightVolume”: “10.5”,

}

Other information:

- User can opt to attach weather information
- Weather information is based coordinate location
- Weather information is obtained from custom build service that provides basic weather information based on coordinates
- The messages with weather information must contain temperature information from the location specified in the request
- The weather service will be provided separately, the server must send the coordinates to the weather service that will provide the weather information to be added to the coordinates

Feature 6: Basic search functionality for observations

URL: https://server.url/search

Request type: GET

Response code for successfully operation: 200

Response code failed operation: 400-499

Header information that must be included in minimum:

- Content-Length: size of the content
- Content-Type: application/json

Authorization: only registered users can perform this operation

User can search with following terms:

- **By nickname**
- **By date range**
- **By observation identification (NOTE: observation id)**
- **Or any combination of nickname, range and/or identification**

Other information

- User can select certain locations (each location has an unique id) as a sightseeing path
- Path is not ordered but rather a "collection" of locations
- locations contain a jsonarray that has a list of locations based on their id (create id for your message), there is no limit for locations other than how many locations a test adds
- query parameters are part of the url (see final lecture for tips)

Feature 7: observation information can be updated

URL: https://server.url/datarecord?id=

Request type: PUT

Response code for successfully operation: 200

Response code failed operation: 400-499

Header information that must be included in minimum:

- Content-Length: size of the content
- Content-Type: application/json

Authorization: only registered users can perform this operation

Data format for posting an update to an existing message:

{

"recordIdentifier" : "Jupiter",

"recordDescription": "Notes about Jupiter during observation",

"recordPayload": "Jupiter great red spot noticed; no changes related to previous observation. EDIT: Corrected language",

"recordRightAscension" : "05h 11m 42s",

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Matti”,

“recordTimeRecieved” : “2024-12-21T07:57:47.123Z”,

“Observatory” :

[

{

“observatoryName”:”Puolivälinkankaan vesitornin observatorio”,

“latitude”: “65.0580507136066”,

“longitude”: “25.469414591526057”,

}

]

}

Where:

- id is an unique integer representing a specific location (it needs to be given by the server when the original location is posted)
- id is used as a query parameter in order to tell the server what message should be replaced (for example localhost/datarecord?id=129)
- id is only supplied by server when messages are retrieved, client never includes the id in the message (only to the url if a specific message needs to be updated)

Example server response for GET

[

{

“id”:”129”,

“recordIdentifier” : “Jupiter”,

“recordDescription”: “Notes about Jupiter during observation”,

“recordPayload”: “Jupiter great red spot noticed; no changes related to previous observation. EDIT: Corrected language”,

“recordRightAscension” : “05h 11m 42s”,

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Matti”,

“recordTimeRecieved” : “2024-12-21T07:57:47.123Z”,

“Observatory” :

[

{

“observatoryName”:”Puolivälinkankaan vesitornin observatorio”,

“latitude”: “65.0580507136066”,

“longitude”: “25.469414591526057”,

}

],

“updatereason”: “N/A”,

“modified”: 2024-1-21T07:57:47.123Z”

}

]

Where:

- updatereason is a string where user can state a reason for the update. If not given by user in original message, server must set the updatereason as “N/A” as default.
- modified is a timestamp when the edit was made

Other information:

- User can send updates to the observations user has sent
    - The modified message must show a timestamp of the latest edit
- In order to identify any record in database, supply an id (integer) with the message to provide the update to a right message
- Whether user is allowed to update the message, the server needs to check if the user (not nickname) has posted the danger message previously. Remember, the nickname is just nickname user chooses for particular message, the real identificatory is the registered user
- User can give a reason for the update or leave the field empty (it still must be included in the message when being updated and when retrieving updated messages from the server)

Feature 8: Decipher observation payload

URL: https://server.url/datarecord?shift=13

Request type: POST
Response code for successfully operation: 200

Response code failed operation: 400-499
Header information that must be included in minimum:

- Content-Length: size of the content
- Content-Type: application/json

Authorization: only registered users can perform this operation
Data format for posting a visit to a site:


```
{
        "recordIdentifier": "Jupiter”,
        "recordDescription": "Notes about Jupiter during observation",
        "recordPayload": "Weövdrb tbrnd brq cööd äädvprq; ää punåtrc bryndrq dä öbrfväec äocrbfndvää",
        "recordRightAscension" : "05h 11m 42s",
        "recordDeclination": "+22° 15' 54\""
}
```


Where shift is the key for the Caesar cipher and recordPayload contains the ciphertext.
Shift is provided in the request URL as a query parameter. The ciphertext should be deciphered using the deciphering service (provided separately, check Moodle). Only the user who posted the message should receive the deciphered text. All other users must receive the ciphertext when getting messages from the server.

Decipher service information:
URL: http://localhost:4002/decipher?shift=13

Request type: POST
Request body example: Weövdrb tbrnd brq cööd äädvprq; ää punåtrc bryndrq dä öbrfväec äocrbfndvää
Response body example: Jupiter great red spot noticed; no changes related to previous observation

Example response for the **user who posted the message**:


```
{
        "recordIdentifier": "Jupiter”,
        "recordDescription": "Notes about Jupiter during observation",
        "recordPayload": "Jupiter great red spot noticed; no changes related to previous observation",
        "recordRightAscension" : "05h 11m 42s",
        "recordDeclination": "+22° 15' 54\"",
        "recordOwner" : "Matti",
        "recordTimeRecieved" : "2024-12-21T07:57:47.123Z"
}
```


Example response for **other users**:


```
{
        "recordIdentifier": "Jupiter”,
```

```
                "recordDescription": "Notes about Jupiter during observation",
                "recordPayload": "Weövdrb tbrnd brq cööd äädvprq; ää punåtrc bryndrq dä öbrfväec äocrbfndvää",
                "recordRightAscension" : "05h 11m 42s",
                "recordDeclination": "+22° 15' 54\"",
                "recordOwner" : "Matti",
                "recordTimeRecieved" : "2024-12-21T07:57:47.123Z"
}
```

Extra Feature 9: Give a summary of payload contents in description

URL: https://server.url/datarecord/aidesc

Request type: POST

Response code for successfully operation: 200

Response code failed operation: 400-499

Header information that must be included in minimum:

Content-Length: size of the content

Content-Type: application/json

Authorization: only registered users can perform this operation

Data format / payload for posting a sightseeing location:

{

“recordIdentifier” : “Jupiter”,

“recordDescription”: “”,

“recordPayload”: “On February 25, 2025, Jupiter's Great Red Spot (GRS) was prominently visible from Earth. The GRS is a massive, high-pressure storm system located in Jupiter's southern hemisphere, rotating counterclockwise. During this time, Jupiter was observable well past midnight, setting after 3 a.m. local time on the 1st and by 1:30 a.m. on the 28th. Telescopic views revealed its dark equatorial belts, four Galilean moons, and occasionally the Great Red Spot. Scopes over 6 inches in diameter provided finer details of Jupiter's dynamic atmosphere, especially during moments of steady seeing. Jupiter's disk spanned 43 arcseconds on February 1 and decreased to 40 arcseconds by the end of the month. Its rapid rotation, completing in under 10 hours, allowed observers to notice cloud motion within 10 to 15 minutes.”,

“recordRightAscension” : “05h 11m 42s”,

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Matti”

}

Response payload:

- N/A

Where:

- recordDescription is initially empty and AI should provide a short summary of recordPayload

Example data format for sending locations back to client:

[

{

“recordIdentifier” : “Jupiter”,

“recordDescription”: “Jupiter's Great Red Spot was clearly visible on Feb 25, 2025.”,

“recordPayload”: “On February 25, 2025, Jupiter's Great Red Spot (GRS) was prominently visible from Earth. The GRS is a massive, high-pressure storm system located in Jupiter's southern hemisphere, rotating counterclockwise. During this time, Jupiter was observable well past midnight, setting after 3 a.m. local time on the 1st and by 1:30 a.m. on the 28th. Telescopic views revealed its dark equatorial belts, four Galilean moons, and occasionally the Great Red Spot. Scopes over 6 inches in diameter provided finer details of Jupiter's dynamic atmosphere, especially during moments of steady seeing. Jupiter's disk spanned 43 arcseconds on February 1 and decreased to 40 arcseconds by the end of the month. Its rapid rotation, completing in under 10 hours, allowed observers to notice cloud motion within 10 to 15 minutes.”,

“recordRightAscension” : “05h 11m 42s”,

“recordDeclination”: “+22° 15' 54””,

“recordOwner” : ”Matti”,

“recordTimeRecieved” : “2024-12-21T07:57:47.123Z”

}

]

AI service information (in gitlab):
URL: http://localhost:8000

Other information:

- Use the course example on how to communicate with the AI (chatgpt / nomicai)
    - You can use your own chatgpt api access if you have any for testing or download the falcon model used in the example from gpt4all
    - Or you can test it by using the ai service in gitlab (it is very slow…) (port 8000)

Additional Quality Improvements

If the software quality is high enough, the final grade of the assignment can be increased by 1. Note that these are evaluated by the teacher and there are no guidelines what is required in minimum. Here are examples of the quality aspects we look for:

- Properly commented code/interfaces
- Usage of unit tests
- Old messages are archieved after certain amount of time (for example after 2 months the messages are removed from the actual database to for example a text file or other database to prevent the database becoming too large affecting the speed of it)
- System backup
- System recovery
- Clean code
- …