Project in the Software Technology Course Specifications of the REST web API and the Command-Line Interface - CLI of the electric vehicle charge management information system

REST web API: General principles The electricity market data management software system you will be developing should support a RESTful Application Programming Interface (REST API) to retrieve the data stored in the Database you will implement. These calls are typical examples of requests from various stakeholders to the System Database, which is considered to meet the implementation requirements of the two common use cases. Base URL The REST API will be available at the following base URL for all tasks: https://localhost:8765/evcharge/api The individual Resources (REST endpoints) that will be available through the API will be accessible through the above base URL, as follows:

{baseURL}/{service}/{path-to-resource}

Where {service} one of the services will be available, as listed below:

For example, the endpoint for retrieving the number of vehicle charges with ID "AB123456" for the month of November 2020 is https://localhost:8765/evcharge/api/SessionsPerEV/AB123456/20201101/20201130.

All results returned by the API will be sorted according to the time field they contain (start time, where there is end time) in ascending order. Data formats

The REST API will support the JSON format (content-type: application / json) and the CSV format (content-type: text / csv). The choice of format will be specified in the application as follows (query parameter):

{baseURL}/{service}/{path-to-resource}?format={json|csv}

If the format parameter is not provided in a request, consider json to be the default value. In any case the character encoding must be UTF8. For example, the previous call with a "csv" data format request is as follows:

https://localhost:8765/
evcharge/api/SessionsPerEV/AB123456/20201101/20201130&format=csv.

User accreditation

To control the access of different stakeholders to the system, the use of the API will require user accreditation. User accounts will be created by the system administrator via the Command Line Interface (CLI), as described below. When calling the API, user access tokens, encoded in the way you deem most compatible with relevant international best practice, should be provided in a dedicated custom HTTP Header.

The custom HTTP header name should be X-OBSERVATORY-AUTH.

Error management

Each call to the API should return the appropriate HTTP Status Codes in the event of an error. In particular, the following error codes will be returned:

400: Bad request = In case the parameters given in a call are not valid (eg empty required field)

401: Not authorized = In case the request is made by an unaccredited user

402: No data = In case the answer to the call is blank

Access and management

Your back-end will support two endpoints for User Login and Logout. Particularly:

-{baseURL} / login: Supports the POST method and receives the username, password parameters of the user coded as "application / x-www-form-urlencoded". In case of successful accreditation of the user, returns a json object with its token: {"token": "FOO"}.

-{baseURL} / logout: Supports the POST method and does not receive parameters (ATTENTION: the user token that must be "logged out" is contained in the custom HTTP header for this purpose, as mentioned above). In case of success, only the status code 200 (empty response body) returns.

Management Endpoints

Your back-end will support the following endpoints, which will be accessible only to users - system administrators:

-{baseURL} / admin / usermod /: username /: password Supports the POST method for adding a new user or changing the password if the user already exists.

-{baseURL} / admin / users /: username Supports the GET method for reading the data of the specific user

-{baseURL} / admin / users /: username Supports the GET method for reading the data of the specific user

This call will replace the automatic upload of data from charging points. The result will return a json object with three numeric fields: SessionsInUploadedFile, SessionsImported, TotalSessionsInDatabase.

Additional (auxiliary) Endpoints

Finally, your back-end will support the following endpoints, which will act as auxiliary for the fully automated control that will be done during the examination of the work:

-{baseURL} / admin / healthcheck: Supports the GET method and confirms end-to-end connectivity between the user and the database. The back-end, that is, will have to check the connectivity to the NW to respond to the request. In case of successful connection the json object: {"status": "OK" επιστρέ is returned, otherwise status "status": "failed"} is returned.

-{baseURL} / admin / resetsessions: Supports the POST method and initializes the charge event table (delete all charge events), as well as initialize the default admin account (username: admin, password: petrol4ever). If successful, the json object: {"status": "OK" επιστρέ is returned, otherwise status "status": "failed"} is returned.

These utility endpoints do not require user accreditation.

System operation

The following are four characteristic endpoints that can be used for the interaction of different stakeholders to the system, during the implementation of the two common use cases (and several more that you can define).

-{baseURL}/SessionsPerPoint/:pointID/:yyyymmdd_from/:yyyymmdd_to A list is returned with the analysis of the charging events (sessions) for a point and period that are given as parameters in the URL. The representation of returned dates must be in the format "YYYY-MM-DD HH: MM: SS"

-
{baseURL}/SessionsPerStation/:stationID/:yyyymmdd_from/:yyyymmdd_to

Returns a list of point-by-point grouping of charging events for a station and period given as parameters to the URL. For each point the number of events and the energy allocated are returned. The representation of returned dates must be in the format "YYYY-MM-DD HH: MM: SS"

-{baseURL}/SessionsPerEV/:vehicleID/:yyyymmdd_from/:yyyymmdd_to A list is returned with the analysis of loading events (sessions) for a vehicle and period that are given as parameters in the URL. The representation of returned dates must be in the format "YYYY-MM-DD HH: MM: SS"

-{baseURL}/SessionsPerProvider/:providerID/:yyyymmdd_from/:yyyymmdd_to Returns a list with the analysis of charging events (sessions) for a power provider and period that are given as parameters in the URL. The representation of returned dates must be in the format "YYYY-MM-DD HH: MM: SS"

Command Line Interface – CLI

General principles CLI specifications include data recovery calls, which are equivalent to those of the REST API, as well as system management calls. The CLI should only be available from the console (command line, ssh) of the system that hosts the application you are about to build. User accounts are only created through the CLI. Data recovery CLI will support JSON and CSV, just like the REST API. The format selection will be specified in a corresponding mandatory parameter as mentioned below. All results returned by the CLI will be sorted by the time they refer to (start charging), in ascending order. The CLI will be called from the command line with calls of the form:

$ ev_groupXX SCOPE --param1 value1 [--param2 value2 ...] --format fff --apikey kkk

Where XX is your team ID and SCOPE as shown in the table below. If no parameters are given, the CLI-supported parameters should be displayed. For example, the call to retrieve the AB123456 vehicle charging events for the month of November is as follows:

$ ev_groupXX SessionsPerEV --vehicle AB123456 --from 20201101 --to 20201130 --format json --apikey ABCD-EFGH-1234

The following parameters per scope should be supported. There is no need to be logged in for healthcheck, resetsessions, login scopes. The user has to be accredited for logout, SessionsPerPoint, SessionsPerStation,

SessionsPerEV, SessionsPerProvider. The --format and --apikey parameters are required for all calls (--format csv json, --apikey XXXX-XXXX-XXXX where X is a character or number).

System management

The admin can basically do everything in addition to what was mentioned above. For management functions (scope: Admin) the following parameters should be supported. --usermod, creates a new user or changes a password of an existent user

--users, show the state of a user

--sessionsupd, add new sessions from a CSV file