Web Service API Specification

(Doc Template!!!)

(*Players and Points Demo App*)

Version	Date	Author	Description
1.0	2012-10-05	Saurabh (rebugged.com) (license: Creative Commons)	Initial draft for *Chilly Recipes App*
1.1	2012-11-17	Saurabh (rebugged.com) (license: Creative Commons) Downloaded from: https://docs.google.com/document/d/1HSQ3Fe77hnthw8hizqvXJU-qGEPHavMkctvCCadkVbY/	Added version number in response. *Chilly Recipes App*
Actually	Only	Some of the original doc remains.	Most errors now by JV
1.2	2017-11-05	Juhani Välimäki	Modifying the rather nice original doc to be an even better **Web Service API template**.
1.3	2018-03-08	Juhani Välimäki	Some improvements already forgotten.
1.4	2018-09-13	Juhani Välimäki	Moving the common parts before individual web service descriptions start. Improving modifiability and readability.
1.5	2018-09-14	Juhani Välimäki	Minor edits

Extra: Start reading by looking at the **Conventions** and the commonly used **HTTP Status Codes**!

Extra: Some security-related Challenges/Questions in client-server, front-back model:

- How to know that the initial request comes from a **valid front-end client app** who is allowed to contact to our back-end services at all? (And not a DoS attacker, for instance).
- How to **authenticate the user**? How to ensure it's really that person and not somebody else pretending to be that user?
- How to know whether **we still communicate with same** authenticated user and there hasn't been a man-in-the-middle attack?
- How to **secure data** going over internet?
- How to allow the authenticated user to access only **his/her data**?
- How to allow the authenticated user to access only **actions** allowed **for his/her role**?
- How to ensure that the data he/she is modifying hasn't been modified by somebody else's **concurrent transaction** in the database?

Glossary

Conventions

- **Client** - Client application = ~ Front-end (Could be e.g. two different kinds, Native mobile app and Web. User might use English or e.g. Finnish!)
- **Status** - HTTP status code of response.
- All the possible responses are listed under ‘Responses’ for each method, first the possible OK responses, then the possible Error responses. Only one of those responses will be returned from the back-end!
- All responses are in JSON format - as 1 JSON root object with possible other objects inside.
- All request parameters are mandatory unless explicitly marked as `[optional]`
- The type of values accepted for a *request* parameter are shown in the values column like this `[10|<any number>]` .The `|` symbol means *OR*. If the parameter is `[optional]`, the default value is shown in blue bold text, as **`10`** is written in `[10|<any number>]`.

HTTP Status Codes

All our status codes are standard HTTP status codes:
Official source: https://tools.ietf.org/html/rfc7231
Unofficial more verbose reading: https://httpstatuses.com/

(1XX - Informal temporary status codes. Usually not to be reacted to. No need to use!)
2XX - Success of some kind
(3XX - Redirecting to another resource on same server or another server.)
4XX - Error occurred in client's part, maybe **request** not correct
5XX - Error occurred in server's side, like e.g. a crash because of our back-end code not able to handle some exception case, or upstream server (like database) not responding

Some errors selected by Juhani on the next page…

HTTP Request Methods: GET, POST, PUT, DELETE, PATCH, etc.

Official source: https://tools.ietf.org/html/rfc7231#section-4.3

Unofficial nicer reading: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

TCP and UDP ports 80, 8080, 443, 22, 3000, 3306, etc.

Official source: https://www.ietf.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

Unofficial nicer reading: http://packetlife.net/media/library/23/common_ports.pdf

HTTP Status Code	Description (=In which situations you could send them in Back-end response)
200	OK (**Generic OK**)
201	Resource created
202	Accepted (Request accepted, and queued for execution)
204	The server successfully processed the request and is not returning any content.
	* * * * * * *
400	Bad request (**Generic Request-related Error**)
401	Unauthorized / Authentication failure
403	Forbidden
404	Resource not found
405	Method Not Allowed
406	Not Acceptable (A bit odd? https://tools.ietf.org/html/rfc7231#page-60)
409	Conflict (E.g. trying to enter a resource which is already in database)
412	Precondition Failed
413	Request Entity Too Large (Payload too large)
	* * * * * * *
500	Internal Server Error (**Generic Server-side Error**)
501	Not Implemented (The requested Web service is not implemented yet)
502	Bad Gateway (Or any other upstream server, like database server or next level web server)
503	Service Unavailable
504	Gateway Timeout (Usually returned automatically by the webserver and not by us)

Common properties and error-handling of our Web Services?

0. Any request before logged in

⇨ Redirected to login GET.
Unless we have public landing / welcome / index / home / default / main page(s)

1. login

Authenticate the user with the system and obtain the auth_token

Server

	URL
`Start of any URI:`	`https:[host]:[port]/api`

Type	Params	Values
HEAD HEAD	api_key auth_key	string string (Required by all other services than **login**)

api_key

`api_key` must be sent with all client requests. The api_key helps the server to validate the request source.
api_key might be:

- unique per client (e.g. per mobile app download or per developer registration)
- or one common key (worst security, and how to update if and when leaked?).

auth_key (string) - all further API calls must have this key in header
// It can be:
1. per session and not changed (not too safe)
2. per session and changed periodically (almost same as 1.)
3. per request and changed after each request (safest, could be implemented with some library. Session used to remember the latest key and who is the corresponding user. Each new auth_key will be passed to the front-end in the response and got back in next request from front-end and will be subsequently checked in back-end. Heavy but safest)
// console.log('Creating Hash with the crypto module');
const hash = crypto.randomBytes(48).toString('hex');

Request Requirements

- **Every single request of ours must send JSON*, even if just {“id”:70001} and use only Http methods GET or POST**

(=Is it okay we decide not to use PUT / DELETE / PATCH etc. verbs at all ???)

*** E.g. [5,7,3,41], [], or {} are valid JSON. The first is a JSON array with four number elements in certain order.**

Response Requirements

- **Every single response of ours must send JSON (either successful data object* or an error object) and HttpStatus code back**

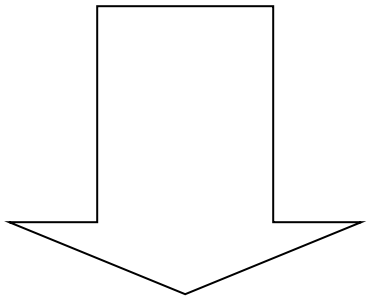
*** E.g. [7] is valid JSON. It’s a JSON array with one number element.**

Common Error cases and their Http Status codes and common error_types

Http Statu	Error JSON object structure	Need to specify also in those web services where used?
400	{"error_type":"required_field_missing", "missing_fields":["firstName","lastName"]} (Copy this to all where used and specify fields)	Yes
401	{"error_type":"invalid_api_key"}	No, common for all
403	{"error_type":"invalid_or_missing_auth_key"}	No, common for all
500	{"error_type":"unspecified_server_error", "inner_error":<errorObjectAsJSONHere>}	No, common for all

(Each Web Service starting from next page)

(Add your text **mainly** to the rest of the document. Starting next page. Suggest edits to texts before this page)



Web Services of our API

0. Any request before logged in

⇨ Redirected to login GET.
Unless we have public landing / welcome / index / home / default / main page(s)

1. login

Authenticate the user with the system and obtain the auth_token

Request

Method	URL
POST	/login
(Internal mapping to)	(name of the internal method if applicable)

Type	Params	Values
HEAD	<COMMON ONES, except no auth_key !!!>	
POST	username	string NOT EMPTY
POST	password	string NOT EMPTY

Responses (The possible Response variations)

Status	Response body (One JSON object, in case of SUCCESS)
200	{} That's a valid empty JSON object **TBD!!!** How aut_key passed to client? In response HEADERS?
Status	**Response body (One JSON object, in case of an ERROR)**
403	{"error_type":"required_field_missing", "missing_fields":["api_key", "user_name","password"]}
401	{"error_type":"incorrect_login_credentials”}

2. Player – Adding a new player

Trying to add one new player to the system (database).

Request

Method	URL
POST	/player/
(Internal mapping to)	(name of the internal method if applicable)

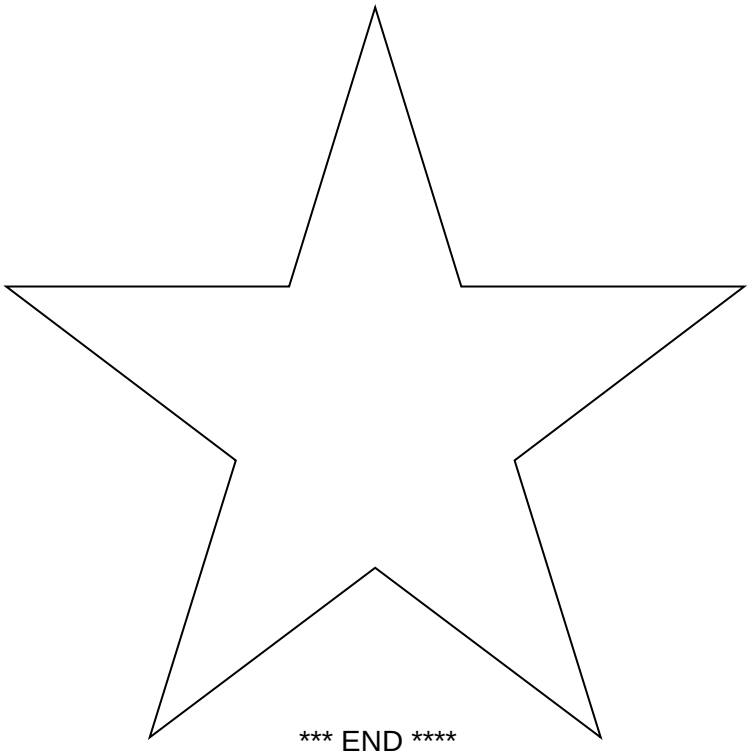
Type	Params	Values
HEAD	<COMMON ONES!!!>	
POST	name	string NOT EMPTY
POST	points	number >= 0

Responses (The possible Response variations)

Status	Response body (One JSON object, in case of SUCCESS)

200	{
	"id":"<id_of_the_added_player>"
	}
	
	(OR in case of **Knex** could be just: **[7]**
	or writing another way: [<id_of_the_added_player>])

Status	**Response body (One JSON object, in case of an ERROR)**
	…missing fields etc…
409	{"error_type":"trying_to_enter_duplicate_row"}
409	{"error_type":"trying_to_enter_duplicate_value"} // if unique constraints
	…you got the idea from the /login, right?…
	…etc…



*** END ****