Crowd Share – Services description

Practical exam JavaScript Applications - 29 July 2014

Crowd Share Server Logic

All of the logic is on the server. The server provides a REST API, enabling login users, register users, posting, or fetching posts.

All the communication between the client and the server is **RESTful**, through **HTTP GET** and **HTTP POST** requests, sent asynchronously as **AJAX** calls from the client, to which the server responds with a **JSON-serialized result**.

On **success**, the server returns the **HTTP status code 200**, **along with a JSON object**.

On **error**, the server returns a 3-digit **HTTP error code** (4xx or 5xx), along with an error description. Error descriptions are in the following form:

Body	`{"Message":"Username already exists","errCode":"ERR_DUP_USR"}`

Where **Message** should be shown to the user and the **errCode** can be used by the client application to process the type of error, but should not be shown to the user.

Each REST service requires a different request format, which will be described, assuming the root for all services is the following URL:

```
http://services-root/
```

For example, if the Login endpoint is described to be found on the URL http://service-url/auth and if the services are deployed on http://localhost:3000 , then the actual URL of the Login endpoint will be http://localhost:3000/auth

User Endpoints

User endpoints provide registration, login and logout. Each application must use the user services in order to begin communication with the server.

Each user has a username and password. The username and password are used for login purposes.

User services share a common root in the form:

```
http://services-root/user/
```

Registration endpoint

The registration endpoint is located at:

`http://services-root/user`

The registration service allows a client to register a user account. This account is later used to access any other services. Requests to the register endpoint must be in the following form:

URL	http://services-root/user	Method: **POST**

Body	{
	"username":"obiwan",
	"authCode":"c79e32c27a1f1241f0da218d57bf15ea9e09e7dc"
	}	

Here **authCode** is the result of a concatenation of the user's username and password, encoded with the SHA1 algorithm. For example, if the user's password is "**secret**" and the username is "**benfranklin**", authCode = **SHA1("benfranklinsecret")** = "**9aab5aedfa56b93dd9e035ff2ecd2215a148e3d9**".

In case of success, the registration service returns an HTTP status code 201 and a JSON object in the form:

Body	`true`

In case of an error, the service returns an HTTP error code, along with an error description as aforementioned.

Login endpoint

The login endpoint is located at:

`http://services-root/auth`

The login service allows a client to enter an existing account, requiring a username and password. The request is in the following form:

URL	`http://services-root/auth`	Method: **POST**

Body	{
	"username":"obiwan",
	"authCode":"c79e32c27a1f1241f0da218d57bf15ea9e09e7dc"
	}	

Here **authCode** is the result of a concatenation of the user's username and password, encoded with the SHA1 algorithm. For example, if the user's password is "**secret**" and the username is "**benfranklin**", authCode = **SHA1("benfranklinsecret")** = "**9aab5aedfa56b93dd9e035ff2ecd2215a148e3d9**".

In case of success, the registration service returns an HTTP status code 200 and a JSON object in the form:

Body	{
	"sessionKey":"172Jp3twchjt8I3DpI3Fdb4t5z7weEQgnDey5HZTDgJEBRp3Yn",
	"username":"obiwan"
	}

The **client should store the returned sessionKey**, as it is necessary to use the other Crowd Share endpoint

In case of an error, the service returns an HTTP error code, along with an error description as aforementioned.

Logout endpoint

The logout endpoint is located at:

```
http://services-root/user
```

The logout service **ends the current user session**, **invalidating the current session key**, disabling access to the other services (except login and register). The user can later login again, to receive a new session key.

Requests to the service **require a session key** to be appended at the end of the service URL – the session key is used to **authenticate the current user**.

URL	http://services-root/user	Method: **PUT**
Body	true	
Headers	The logout service requires an additional header (X-SessionKey) with value that is equal to the session key of the logged-in user	

In case of success, the service returns a body 'true' with an HTTP status code 200.

In case of an error, the service returns an HTTP error code, along with an error description as aforementioned.

Posts Endpoints

Posts endpoints provide functionality for visitors to view the posts or create new posts

Get posts

The "**get posts**" endpoint is located at:

```
http://services-root/post
```

The get posts endpoint returns posts from the server.

The posts can be filtered either by username, search pattern in the title and in the body of the post, or both. The filtering is performed using optional query parameters: user and/or pattern. The query parameters are appended at the end of the endpoint.

The endpoints variations:

URL	Description
`http://services-root/post`	Returns all posts
`http://services-root/post?`**`user=minkov`**	Returns only the posts from user Minkov. The username provided is case-insensitive
`http://services-root/post?`**`pattern=Again%20Ipsum`**	Return only the posts that contain the pattern "Again%20Ipsum" ("Again Ipsum") in either their title or their body. The pattern is case-insensitive.
`http://services-root/post?`**`pattern=lorem%20ipsum&user=minkov`**	Return only the posts that contain the pattern "Again%20Ipsum" ("Again Ipsum") in either their title or their body, and are from user Minkov. The pattern and the username are case-insensitive.

URL	`http://services-root/posts`	Method: **GET**
Body	*(GET request => empty)*	

In case of success, the service returns a response, containing the posts, with an HTTP status code 200:

Body

```
[{
  "id": 1,
  "title": "Lorem ipsum",
  "body": "Lorem ipsum, lorem ipsum, and again lorem, lorem, loreeeeeeem",
  "postDate": "2014-07-28T05:05:33.853Z",
  "user": {
    "id": 1,
    "username": "Minkov"
  }
}, {
  "id": 2,
  "title": "Ipsum #2",
  "body": "Lorem ipsum, lorem ipsum, and again lorem, lorem, loreeeeeeem",
  "postDate": "2014-07-28T05:06:04.410Z",
  "user": {
    "id": 2,
    "username": "Yoda"
  }
}, {
```

	<pre> "id": 3,
	  "title": "Again Ipsum",
	  "body": "Lorem ipsum, lorem ipsum, and again lorem, lorem, loreeeeeeem",
	  "postDate": "2014-07-28T05:06:15.090Z",
	  "user": {
	  "id": 1,
	  "username": "Minkov"
	  }
	}]</pre>

In case of an error, the service returns an HTTP error code, along with an error description as aforementioned.

Create post endpoint

The "**create post**" endpoint is located at:

```
http://services-root/post
```

Requests to the service **require a session key** to be provided as value to the HTTP header "X-SessionKey" – the session key is used to **authenticate the current user**.

URL	http://services-root/post	Method: **POST**

Body	<pre>{
	 "title": "To Lightsabers! ",
	 "body": "Dear sith friends, it is about time to crush these puny jedi and their troppers. They are powerful, but we have Dark Side with us! "
	}</pre>	

Headers	The create post endpoint requires an additional header (X-SessionKey) with value that is equal to the session key of the logged-in user	

In case of success, the service returns a response with the created post with an HTTP status code 200.

In case of an error, the service returns an HTTP error code, along with an error description as aforementioned.