Crowd Share – Web Client

Practical exam JavaScript Applications - 29 July 2014

You are given server logic in the form of a REST API in Node.js, which implements the Crowd Share. Your task is to develop a client-side application using HTML, CSS and JavaScript (+jQuery), that uses (consumes) these services, to allow users to share their posts, using their web browser.

The server is started through the app.js script from Node (you need first to install the dependencies) or by starting the commands start-server.txt file.

The server is available locally on your computer or online here: http://jsapps.bgcoder.com/

Crowd Share Description

Visitors must register in order to post to the Crowd Share application.

The application has the following workflow:

1. Visitors enter the application
    a. They can view all the posts, even if not logged-in
    b. They can register or login a user
2. Visitors can view all posts, sorted
    a. The visitor chooses how the posts to be sorted:
        - By date or by title (alphabetically)
        - Ascending or descending
    b. The visitor can see the posts in pages. Each page contains N posts
        - N is provided by the visitor
        - Visitors can navigate through the pages (previous or next)
3. Visitors can register a user
    a. Providing a username and a password
4. Visitors can login into the application
    a. Providing a username and a password
    b. They receive an unique session key that is used to authenticate to the application
5. Logged-in users can logout from the application
    a. Providing their session key (received after the login)
6. Registered users can post into the Crowd Share
    a. Providing a title, body and a session key (received after the login)

Task description

Using the REST API, implement the Crowd Share application, following the requirements:

Implementation Requirements

Use all the endpoints from the REST API. Full description can be found in **Crowd-Share-Services.docx/ Crowd-Share-Services.pdf** file.

Method	Endpoint	Brief description
POST	/user	Registers an user. Needs a username and authentication code
POST	/auth	Logs-in an user. Needs a username and authentication code. Returns a session key.
PUT	/user	Logs-out an user. Needs a session key
POST	/post	Creates a new post. Needs a session key
GET	/post	Fetch posts from server. Can filter posts either by User or by search pattern

Application Structure Requirements

Divide your application into reusable and with-tests-in-mind modules:

1. Module for generating UI
2. Module for access to the REST API
3. Module for application control flow

Technical requirements

You must use the following libraries:

- Underscore.js
- Require.js
- A library that provides promises (ex. Q or RSVP)
- Mocha and Chai

Testing requirements

Your unit tests must be done using Mocha and Chai. All tests must use fake posts data, without actual HTTP requests to the REST API.

- Test the sort behavior
    - Without posts
    - With posts
    - Both by title and by date

- Test the paging behavior
    - Without posts
    - With posts

User Interface requirements

You are free to use any UI framework or UI helper framework of your choice. Just create a usable user interface, where users don't wonder where to click to make a thing happen.

Some, but not all of the frameworks are:

- CSS framework like Twitter Bootstrap
- UI components framework like Telerik KendoUI (only the UI components) or jQuery UI
- Template framework like Handlebars.js or Mustache.js
- Routing framework like Sammy.js (only the routing)

You are not allowed to use any framework that provides an MV* architecture like:

- AngularJS
- Backbone.js
- Ember.js
- KendoUI
- Sammy.js
- Knockout.js
- Knockback.js