Università
della
Svizzera
italiana

**Facoltà
di scienze
informatiche**

Web Atelier
Prof. Cesare Pautasso
Masiar Babazadeh
Vincenzo Ferme
Vasileios Triglianos

HTML5 JavaScript APIs Exercises – 02.10.2015 / due 07.10.2015

Before staring:
Get and extract the exercise skeleton from Moodle. The skeleton has the following structure:

```
atelierbeats
|-- js
|   |-- app.js
|   |-- handson.js
|   |-- library.js
|   `-- model.js
|-- resources
|   |-- ...
|-- css
|   |-- ...
|-- images
|   |-- ...
|-- fonts
|   |-- ...
|-- tracks
|   |-- ...
|-- album.html
|-- artist.html
|-- library.html
|-- login.html
|-- signup.html
`-- test.html
```

Deployment:
The completed exercise should be deployed on the `atelier.inf.usi.ch` Web server, under:
`http://atelier.inf.usi.ch/~<account>/wa-ex3/`

Deliverables:
Submit on Moodle your website as a zipped archive that contains:

1. your views (HTML pages) in the root directory. Do not move `test.html`
2. your JavaScript files under the `js` folder. At minimum `app.js`, `model.js` and `library.js` should be there.
3. any static assets (CSS, images, fonts. **NO audio files**)
4. a readme file with your name, the exercises you implemented and a link to the online version of the web application, which should be also deployed on `atelier.inf.usi.ch`

Constraint:
In last week's Exercise you implemented the constructors of the six objects required in AtelierBeats. With the same constructors you implemented we built a collection of instances of those objects, you will find them in `library.js`. Given the data in the provided library you are required to implement all the functionalities described in this exercise. Moreover the file *`model.js`* provides a **simplified** behavior-less version of the constructors of last exercise, do **NOT** touch those constructors since they are used in `test.html` to test your implementation.

Tip: There are some very small differences in the objects you created last week and the ones given in this exercise, e.g. instead of a timestamp *`date_created`* contains a long string. Do not worry; you won't need to use it in this exercise.

Exercise 1. Render HTML views from JSON data (30 points)

In this exercise you will render views dynamically, by using the data from the given object models.

In `app.js` you can find a `window.onload` handler. Inside the handler you have to implement the logic to render the `JSON` data (which are exposed in a global variable in `library.js`)

Render the model data from `library.js` in the *library*, *artists* and *albums* view. This means that instead of displaying static HTML, you will iterate through every object to convert its content into HTML.

1. The track objects are shown in the library view (10 points)
2. Every artist in the collection is listed in the artists view with its corresponding picture (10 points)
3. Every album is shown in the albums view with its corresponding cover picture (10 points)

Tip: Since tracks reference artists and albums by _id, you will have to resolve these first in order to populate the library view with the name of the artist and album.

Tip: Tracks may have more than one playlist or album inside the `collections` field, you are allowed to return only the first referenced album you find.

Tip: Since all pages are going to use the same `app.js`, it will be convenient to add a different class in the `body` of your HTML pages and based on this class render the correct view. For example the `body` of the library view can have the class `library`

Exercise 2. Live Search (20 points)

1. Implement the body of the `fuzzyFind` function in `model.js. fuzzyFind` takes as arguments an Array of objects, the property of each object to search and a term to search that property for. If the value of the property **contains** the term, ignoring case, the function returns an array with the corresponding objects.

2. Use `fuzzyFind` to implement live search on your library so that the content of the page is dynamically updated with the result of the search. A user should be able to type the name of the track (or a substring of it) on the search box and see the filtered results rendered on the track list of the library.

Exercise 3. Basic Music Player (20 points)

Now it's time to have a working music player. Implement the function `setupPlayer` in `app.js`.
1. Create an `audio` element, set its `src` attribute to the file property of your first track and append it to the DOM.
2. Make your play\pause work, that is, when the user clicks "play" your player should play its track and the button should switch to the pause icon and vice versa
3. Display the elapsed time (in `mm:ss` format) of a track using the `currentTime` property. This should be updated as the player is playing a track.

   Tip: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_HTML5_audio_and_video

Exercise 4. Render playlist content (10 points)

When a user clicks on a playlist item from the menu, the Content of the library view should display the tracks of the playlist using the same style as for the tracks of the library view. You don't have to create a playlist HTML page, do this dynamically by injecting the content in the tracklist container of the existing library.html page.

Exercise 5. Playlist canvas visualization (10 points)

1. Open library.html and locate section playlistHeader.
2. This section is initially hidden. Whenever a playlist is clicked, the section will be visible and the canvas will show a mosaic of the artworks of the albums associated to the tracks in the playlist. The mosaic shows artworks in a 2x2 grid. If the playlist is composed by tracks that spawn in more than four albums, just draw the first four artwork you find, if the playlist is composed by tracks that spawn in less than four album, then draw as many artworks as possible.

Exercise 6. Persistent Playlists (10 points)

a) Create new playlist button.
Write a handler function for the "New Playlist" button. Clicking on the button should create a new playlist and make it persistent in `localStorage` (that is, by reloading the page, the user should be able to see the new playlist he created).
The playlist object should be created using the `playlist` constructor in model.js, and saved using the `savePlaylist` function, which you have to implement. For more details see the jsdoc documentation above the function declarations.

b) Load playlists from storage and display them.
When the page is loaded, check the `localStorage` and if it contains stored playlists, these should be displayed on the corresponding menu.

Exercise 7. Editable Playlist Names (Extra 10 points)

Add a small `"edit"` icon next to each playlist. When clicked it should allow the user to edit the name of the playlist. When clicked again it should save the name both in the `localStorage` and in the view. If the user types ESC or the name of the playlist is empty, the last not empty stored name should be preserved.

Exercise 8. Add songs to the playlist (Extra 15 points)

Implement a feature that lets you drag songs from your library into a playlist using drag-and-drop. When a user drags a track from the library on top of playlist item on the menu, the `_id` of the song should be appended to tracks of the playlist. The content of the playlist should be saved to `localStorage`.

Exercise 9. Make the player fully functional (Extra 25 points)

The music player is very important for the functionality of your web streaming service. In this exercise we're going to add the rest of the features that are expected of a music player. You have to implement the following:

a) When the player (audio element) is playing it should update the progress bar to show the current position on the song
b) When the progressbar is clicked it should update itself and seek the player's `currentTime`
c) Display the total time time (in `mm:ss` format) of the current track using the duration property of the audio element. (Event `loadedmetadata` can be useful to identify when the duration is available for read)
d) Implement a volume control bar to control the volume property. When the volume bar is clicked it should update the audio element's `volume`.