BoomBoom Video App Tutorial | Sencha Touch

# Foreword

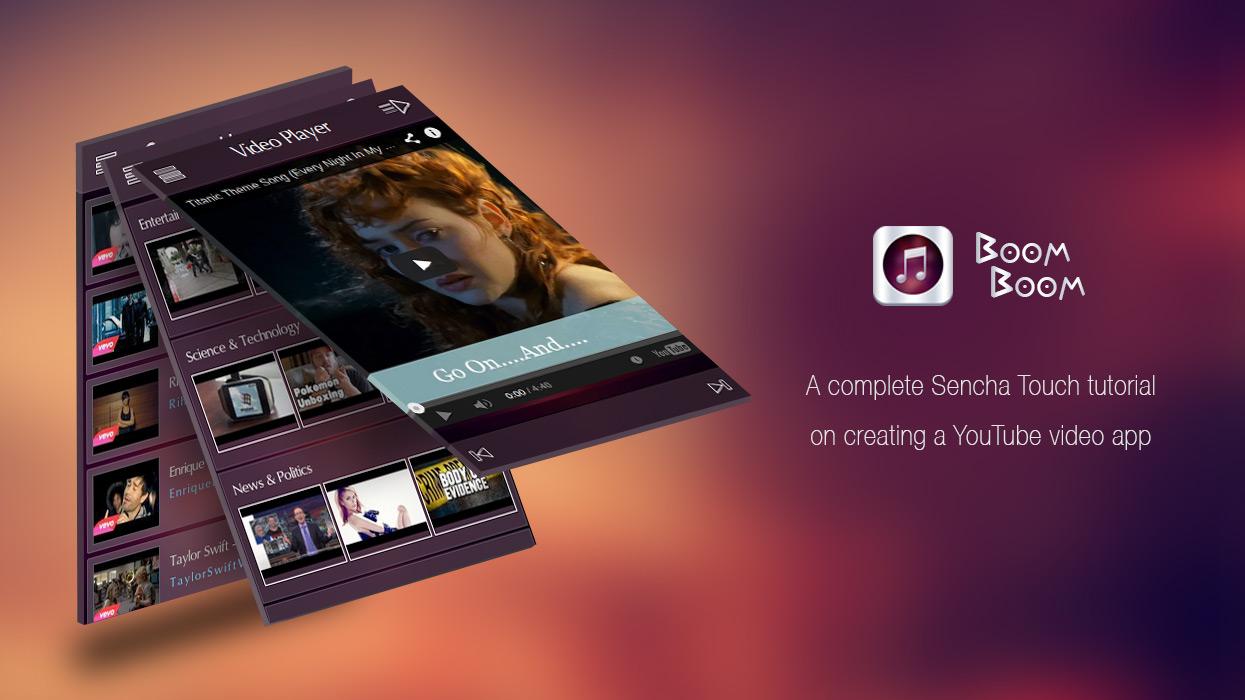
# It’s been over 5 years since we first started working on Sencha Touch. We were one of the early adopters of Sencha Touch when the 0.90 beta version was first released in July, 2010. We immediately fell in love with the library because it shared the same application architecture of ExtJS - a technology which was very popular at the time for providing complete UI based web solutions. In last 5 years, we have developed [numerous Sencha Touch applications](http://innofied.com/work-tag/sencha/) for multiple large scale clients such as IMD Business School, Max International and so forth. On top of that, we have been assisting the Sencha Community regularly, by providing useful resources. We keep sharing useful posts on different aspects of Sencha Touch on [Innofied blog](http://www.innofied.com/blog/). Our [Locator app Tutorial](http://code.tutsplus.com/tutorials/create-a-location-aware-site-with-sencha-touch--mobile-17301) on [Tutsplus+](http://tutsplus.com/authors/swarnendu-de) is extremely popular and is being received very well. BoomBoom Video App tutorial is the second thorough tutorial that we are going to make available for the beginner level developers. Having a large team of expert Sencha developers at Innofied, we decided to come up with a [Sencha Tutorial website](http://www.senchatutorials.org) where our Sencha team as well as other expert Sencha developers will contribute their ideas and make it a great resource for only Sencha based contents. We already spoke with many Sencha developers and all are eager to write for senchatutorials.org. Hope to make this website an amazing learning platform for every Sencha developer; and the BoomBoom Video App tutorial is our first step towards it.

# 



# Swarnendu De Co-founder, Director, Innofied

# Introduction



Developing Sencha applications is quite tricky for beginners. Unlike a number of libraries like Backbone.js or JQuery Mobile, Sencha Touch has a stiff learning curve. A beginner must know the basics of Sencha, its application architecture, main components and SASS/COMPASS to start developing a mobile app. Even then, the developer will need to test the app in different platforms (iOS, Android, Windows Mobile) and learn how to create a native build to upload to respective app stores. This makes it a lengthy process.

Among all the Sencha tutorials available online, only few are actually robust and informative enough to provide a complete idea of designing and developing an app from scratch to complete. Moreover, almost none of them will tell you the benefits of a great app design and show you how to implement that design. In this tutorial, we are going to follow all these concepts and develop a simple YouTube video app by utilizing the YouTube API. The basic idea will be to search videos, play them and add them to a playlist. That’s all. Notable features include,

* Maintaining LocalStorage to save the state of the playlist. Whenever user opens the app, they will be able to see the playlist.
* Having a dashboard to show the featured videos based on categories like Entertainment, Science & Technology etc.

You need to have a basic understanding of HTML, CSS and Sencha to make full use of this tutorial. After completing this tutorial you will learn:

1. How to structure a complete Sencha application
2. How to create wireframes and what type of designs makes an app look cool
3. Which Sencha programming standards you should follow
4. What user experience does the app offer

Note: If you are new to Sencha, then we advise you to finish up the Sencha Guide at sencha.com first.

Now, let’s get started.

Table of Contents

[Foreword](#_5uqcjgmk2vd4)

[Introduction](#_jc7yij23dvj5)

[Takeaways](#_nyzj3zolasce)

[Objectives](#_g2eg0jbei0qw)

[API](#_4yn2wesgsral)

[Features of BoomBoom explained for you!!!](#_oxg0dgja71q)

[Categorized video lists](#_j14lu4uhfjon)

[Instant video search](#_xxsuk8w4uv2q)

[Adding videos of your choice to playlist with benefits of local storage](#_tdiya94di39f)

[Local Storage](#_8o2tafgvw4ej)

[Playing videos from different views](#_b56920ns0r6t)

[Removing videos from playlist just by a swipe](#_2yx2vr5t7kxk)

[Routing in BoomBoom](#_qwv748lob323)

[Some technicalities to enhance our app](#_xh6wdhfulyi9)

[Slide Menu](#_lg35fbq1xyci)

[Sliding playlist attached to the video player](#_m4djjnwz4rcm)

[Featured list and custom data view](#_sppwft62yt6p)

[Activating navigation menu list according to the card in view](#_uv6gbcvkg77l)

[Utility file](#_nwvov9p51if9)

[Language file](#_q2zdy75h39jm)

[Some thoughts for next version](#_8qkzxbsrb49t)

# Objectives

There are some universal objectives that every app should meet. What are they?

1. App must be user friendly and easy to navigate
2. The functionalities should be simple yet attractive
3. The design should be appealing to the user

With these objectives in mind, we will jump into development. While working with any mobile app, we at Innofied, follow a number of steps. Let’s check what they are.

# Step 1: Creating the app wireframes

Creating wireframes is a vital step before proceeding with the design of the app. In this process, both the developer and designer should unite their minds to come up with the best flow of the design. However a developer can independently create such wireframes, before starting with the development process. In our case, we used mockflow.com for this purpose. There are multiple wireframing tools like [Balsamiq](https://balsamiq.com/), [MockFlow](http://www.mockflow.com/) or [JustInMind](http://www.justinmind.com/) (free tool) which you can use to create such mock ups. Following are the wireframes we created for the app:



1. Home/Dashboard 2. Show buttons on tap

In the home page, we are going to have a group of categories and videos from each category will be displayed in a horizontal list (Fig. 1). Clicking on any of the videos will show a small panel below it with two buttons: *Play* and *Add to Playlist* (Fig. 2). Clicking on the same video again will hide this panel.

|  |  |
| --- | --- |
| Page_2.png  3. Side navigation | Page_3.png  4. Search videos |
|  |  |

We will keep a simple slide navigation (Fig. 3) instead of a tab panel. This is because, such apps do not need constant navigation to different pages and having a tab panel may just waste some vertical space. Instead, a slide navigation is easy to implement and will be made available from all the pages.

Search videos screen (Fig. 4) is quite simple, with a search field on top and the search results below that. Clicking on a video works just as it does on Home page.



5. Playlist 6. Video Player

We made the Playlist (Fig. 5) a right side slide panel instead of a separate page. The benefit is, the user will not need to move away from the video player page if he wants to select another item from the playlist. You can remove a video from the playlist by sliding that list item - which displays a *Delete* option. Else, you can clear the complete playlist with the *Clear All* button.

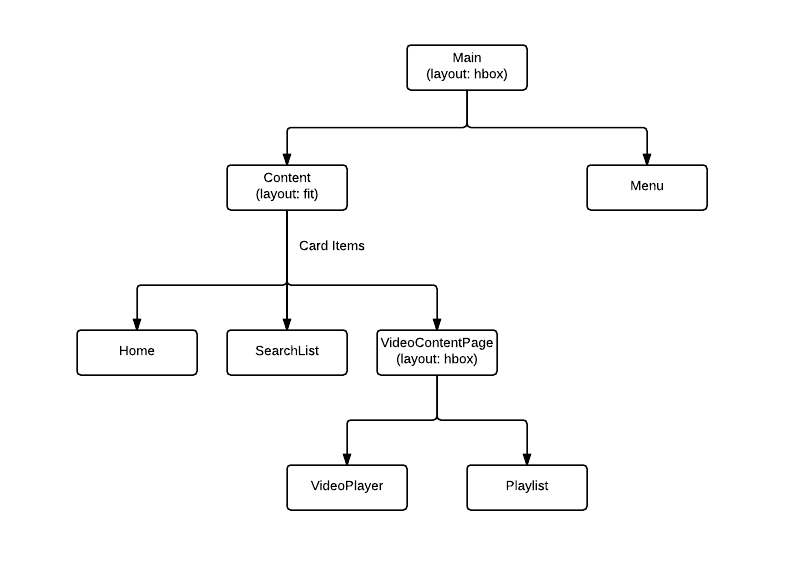
The Video Player screen (Fig. 6) is the most basic to design. We included a previous and next button, to help to navigate through the playlist items.

# Step 2: Start designing

Since we are dealing mostly with the development, we won’t dive too deep into the designing process. You can see individual page designs as we start describing individual page functionality.

Step 3: Creating the app architecture

|  |  |
| --- | --- |
| structure.1.png | We believe, this is the single most important step if you are going to start developing a Sencha application. Not many developers follow this practice and we cannot describe enough how critically important this is for app development. To create the app architecture, we finalize the following things before starting development:   1. The layout we are going to follow in the app 2. All the views, models, stores and controllers 3. Utility methods   The snapshot on the left gives a clear view of the folder structure and the files we used in this app.  We are going to discuss each view in detail along with the models, stores and controllers in the following sections. The number of views are more than the pages that we discussed. It is because we are going to have a number of custom components which will be reused in the major views. Now, let’s see the layout we are going to follow in the app and the reasons for doing so. |



## Why this layout?

In fact, this is quite a simple layout for such a small application. The most important point is to have Hbox layout for *Main* view and *VideoContentPage* view. These two views will have the slide panels and hence Hbox layout is required for these views. The *Content* page will contain the other pages and hence it will need a Card layout.

## Views

Here are the details of the view files:

1. **Main.js:** Parent of all the other views
2. **Menu.js:** Displays the options that users can navigate to.
3. **Content.js:** Container of the main pages of our app
4. **Home.js:** Lists the popular Youtube videos under each category
5. **SearchList.js:** Provides a search field and displays the search results
6. **VideoContent.js:** Parent of VideoPlayer and Playlist page
7. **VideoPlayer.js:** Container for playing the video
8. **Playlist.js:** List of videos to be played
9. **CustomDataview.js:** Parent for any dataview to allow proper horizontal and vertical scrolling
10. **FeaturedList.js:** Horizontal lists for the Home page
11. **VideoOptions.js:** Opens a container when a user clicks on a video. This view allows the user to either play the video or add the video to the playlist.
12. **ListContainer.js:** Contains all the lists that are shown in the home page

## Controllers

Next, we move to the controllers. All the functionalities that you will find in the app are actually implemented using controllers. Don’t just go through these controllers, take your time and try to understand how we have implemented the functions.

Two controllers will be enough for such a small app:

1. **App.js**: Controls the overall functionality of the app
2. **Player.js**: Deals with the functions related mainly to our video player

**Stores**

3 different stores are used in BoomBoom:

* **Videos.js**: Used for the home page
* **SearchListVideos.js**: Used for search page
* **PlaylistVideos.js**: Used for playlist

## Models

We are going to use a single model called **Video.js** for our project.

That is all about the structure and architecture of BoomBoom. It is advised to decide the architecture of the project before you start anything. If necessary you can always add new files to the existing architecture. Since, we have wrapped up the file structure, we can now look into the YouTube data API that we are going to use in our app.

# Step 4: Understanding YouTube data API

We will mainly use two APIs here - the Search API and the Video list API.







### A. [youtube.search.list](https://console.developers.google.com/project/buoyant-dynamo-711/apiui/api/youtube/method/youtube.search.list?authuser=0)

This returns a collection of search results that match the query parameters specified in the API request. By default, a search result set identifies matching video, channel, and playlist resources, but you can also configure queries to retrieve only a specific type of resource. Find the complete details here at <https://developers.google.com/youtube/v3/docs/search/list>

### B. [youtube.videos.list](https://console.developers.google.com/project/buoyant-dynamo-711/apiui/api/youtube/method/youtube.videos.list?authuser=0)

This returns a list of videos that match the API request parameters. We are going to use this to get category wise videos. Find the API details here at <https://developers.google.com/youtube/v3/docs/videos/list>

To use the [youtube.videos.list](https://console.developers.google.com/project/buoyant-dynamo-711/apiui/api/youtube/method/youtube.videos.list?authuser=0) API we have just made an AJAX call with some parameters, as offered by the API and wrote some actions on request of success or failure. In our code we have set ‘useDefaultXhrHeader’ to ‘false’ for supporting cross origin requests. If you are unfamiliar with the term cross-origin requests you should have a look at [Cross Origin Requests](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS), [Default Xhr Header](http://docs.sencha.com/touch/2.4.0/apidocs/#!/api/Ext.data.Connection-cfg-useDefaultXhrHeader).

Now that the API is ready, we can move on to the features.

# Step 5: Start Development

Now that all the basic requirements are done, we can jump into development. The idea here is to not discuss every small detail of developing the app. Instead we will deal with the process of creating major features and the effective solutions that we had to utilize for them.

* [Categorized video lists](#_6ravldatb11q)
* [Instant video search](#_d73zstalmidp)
* [Adding videos of your choice to playlist with benefits of local storage](#_7x74kot3q2j2)
* [Playing videos from different views](#_qag7psp50111)
* [Removing videos from playlist just by a swipe](#_va87lxy2j1e3)
* [Custom-made video player controls](#_hv66f029bqbb)
* [Routing in BoomBoom](#_qwv748lob323)

|  |
| --- |
| [Some technicalities to enhance the app](#_xh6wdhfulyi9)   * [Slide Menu](#_lg35fbq1xyci) * [Sliding playlist attached to the video player](#_m4djjnwz4rcm) * [Featured list and custom data view](#_sppwft62yt6p) * [Activating navigation menu list according to the card in view](#_uv6gbcvkg77l) * [Utility file](#_nwvov9p51if9) * [Language file](#_q2zdy75h39jm) |

Too much information to grasp? Don't worry, we will be explaining each and every point mentioned above, in a short while.

## Categorized video lists

Our Home View contains horizontally scrollable lists. Each list makes an API request to [youtube.videos.list](https://console.developers.google.com/project/buoyant-dynamo-711/apiui/api/youtube/method/youtube.videos.list?authuser=0) and fetches data according to the category specified by us.



Because there are multiple horizontal lists for different categories, we tried to avoid repeating the same thing again. Instead, we have written two functions in App.js controller to load all the lists in the home view. Before you look into the functions, have a look at the reference and control in App.js which is required to load the contents of the home view. Let’s see how to reuse the views and populate the home view dynamically.

refs: {

// views

home: 'home',

homeContent: 'container[name="home\_content"]',

...

},

control:{

home: {

initialize: 'loadHomePageContent',

}

..

}

Now, let’s load all the feature lists one by one. First we will have a config object ready that will tell us the categories that are to be shown in the Home screen.

/\*

\* Content of a page is loaded on page activate event

\*/

loadHomePageContent: function() {

var me = this,

// config array to load as many categorised lists as required

content = {

entertainment: {

id: 24,

label: 'Entertainment'

},

scitech: {

id: 28,

label: 'Science & Technology'

},

news: {

id: 25,

label: 'News'

},

sports: {

id: 17,

label: 'Sports'

},

gaming: {

id: 20,

label: 'Gaming'

},

music: {

id: 10,

label: 'Music'

},

trailors: {

id: 44,

label: 'Trailors'

},

petsAndAnimals: {

id: 15,

label: 'Pets & Animals'

},

travelandEvents: {

id: 19,

label: 'Travel & Events'

},

filmAndAnimation: {

id: 1,

label: 'Film & Animation'

},

autosAndVehicles: {

id: 2,

label: 'Autos & Vehicles'

}

},

listContainer,

list,

videoOptions,

homeContent = me.getHomeContent(),

home = me.getHome();

// Loads all the lists input params are categoryId and the list

for (var item in content) {

var category = content[item];

listContainer = Ext.create('BoomBoom.view.ListContainer');

list = Ext.create('BoomBoom.view.FeaturedList');

// assigning lists parent as home view

list.config.parentPanel = home;

me.loadFeaturedLists(category.id, list);

listContainer.add(list);

// setting label in the listContainer view

listContainer.getItems().items[0].setData(category);

homeContent.add(listContainer);

}

},

/\*

\* Loads different featured list based on parameters

\*/

loadFeaturedLists: function(categoryId, list) {

var me = this;

Ext.Ajax.request({

// api url

url: me.util.api.videoUrl,

method: 'GET',

// To support cross-origin requests

useDefaultXhrHeader: false,

// Passing Parameters

params: {

part: 'id,snippet,contentDetails,player',

chart: 'mostPopular',

videoCategoryId: categoryId,

maxResults: 10,

key: me.util.api.key

},

success: function(response, request) {

var data = Ext.decode(response.responseText),

videostore = Ext.create('BoomBoom.store.Videos');

videostore.setData(data.items);

list.setStore(videostore);

},

failure: function(response, request) {

me.util.failedRequest(response.statusText);

}

});

},

failedRequest: function(errorText) {

var error = errorText || 'Connection Error';

Ext.Msg.alert('Error', error, Ext.emptyFn);

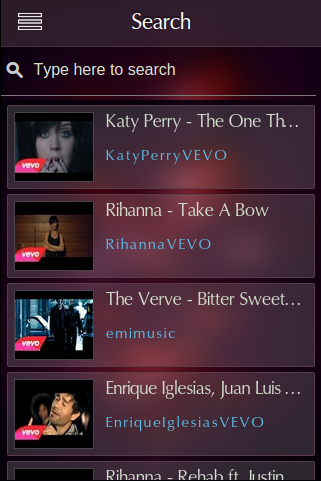
},

The above function gives a message alert when invoked. It alerts the user about why the API request failed.

This is all to understand how to load categorized lists better. Next,we will discuss about our ‘Instant Video Search’.

## Instant video search

One of the objectives of our app is to search Youtube videos. Users can search a particular video by typing its title or the artist name or any relevant query in the search field. Related videos will automatically appear in the result. This is how the results look in the original design.



/\*

\* delay 1 second before showing search results

\*/

delaySearchResult: function(field) {

var me = this;

if (me.searchTask) {

me.searchTask.cancel();

}

me.searchTask = Ext.create('Ext.util.DelayedTask', function(){

me.showSearchResult(field);

});

//Search list will be shown after 1 second of last time key pressing

in the search field

// This process actually reduces the number of requests from api

me.searchTask.delay(1000);

}

/\*

\* Fetch searched videos from api using Ajax call and show in list

\*/

showSearchResult: function(field) {

var me = this,

//the text typed in the search field

query = field.getValue();

Ext.Ajax.request({

//getting url from util file

url: me.util.api.searchUrl,

method: 'GET',

params: {

part: 'snippet',

q: query,

regionCode: 'IN',

maxResults: 30,

key: me.util.api.key

},

success: function(response, request) {

// Format the response via built in function Ext.decode()

var data = Ext.decode(response.responseText);

//Setting data of the store 'SearchListVideos'

Ext.getStore('SearchListVideos').setData(data.items);

},

failure: function(response) {

//Getting error message from response of api

me.util.failedRequest(response.statusText);

}

});

}

## Adding videos of your choice to playlist

Videos can be added to playlist from two views, one is home view and the other is from searchlist view.

We will first discuss about adding videos from searchlist view.

Once you select a video, you get the option to play it immediately or add it to a desired playlist.

|  |  |
| --- | --- |
| homeoptions.png | searchlist.png |

Now, if you tap on the play button, the corresponding video will start playing immediately, while automatically getting added to the playlist. Similarly, on tapping the ‘Add to Playlist’ button, that video gets added to the playlist, as the last in order. We wanted to have the playlist accessible even if the app is closed. Hence, we used HTML5 Localstorage for saving the playlist locally.

/\*

\* Add tapped item(video) to store and playlist

\*/

addVideoToPlaylist: function(recordData, isPlayBtn) {

var me = this,

id = recordData.id,

videoId = recordData.id.videoId,

playlistStore = Ext.getStore('PlaylistVideos'),

exist = playlistStore.find("vid", (videoId || id));

if (exist >= 0) {

if (!isPlayBtn) {

me.showToast("Already exists in playlist");

}

// this returned value will be used as an index to select the video in playlist

return exist;

} else {

Ext.Ajax.request({

url: me.util.api.videoUrl,

method: 'GET',

useDefaultXhrHeader: false,

params: {

part: 'player,snippet,contentDetails',

id: (videoId ? videoId : id),

key: me.util.api.key

},

success: function(response) {

me.addVideoOnSuccess(response, recordData, isPlayBtn);

if (!isPlayBtn) {

me.showToast("Added to playlist");

}

},

failure: function(response) {

me.util.failedRequest(response.statusText);

}

});

}

},

/\*

\* Add video to playlist store and local store on success

\*/

addVideoOnSuccess: function(response, recordData, isPlayBtn) {

var results = Ext.decode(response.responseText),

data = results.items[0],

me = this,

localStore,

addedVideo,

playlistStore = Ext.getStore('PlaylistVideos');

recordData.duration = data.contentDetails.duration;

recordData.videoLink = data.player.embedHtml;

localStore = {

title: recordData.title,

channelTitle: recordData.channelTitle || 'Anonymous',

id: recordData.id,

vid: recordData.id.videoId || recordData.id,

thumbnail: recordData.thumbnail,

duration: recordData.duration

};

addedVideo = playlistStore.add(localStore);

me.getPlaylistClearButton().setDisabled(false);

playlistStore.sync();

},

The above function does the work of adding the video item tapped to the playlist. To avoid repetition of adding the same item again and again to the playlist we had kept a check for results that checks the ‘vid’ field value with the current item videoId or id value. If matched it sets the status flag and returns it.

if (playlistStore.findRecord("vid", (videoId || id))) {

}

We have used a component called ‘Sencha Toast’ while adding a video to the Playlist

showToast: function(toastMessage) {

Ext.toast({

message: toastMessage,

margin: '90% 0% 0%',

timeout: 400,

autoDestroy:true,

showAnimation: {

type: 'fadeIn',

duration: 250,

easing: 'ease-out'

},

hideAnimation: {

type: 'fadeOut',

duration: 250,

easing: 'ease-out'

}

});

},

|  |  |
| --- | --- |
| toast.png | toast2.png |

Coming on to the Home view. When an item in any featured list is tapped, it shows a VideoOptions view accompanied by two buttons 1) **Play** and 2) **Add to Playlist**. What happens when the user taps on play button is explained in next section. Here, we will discuss the latter. When the user taps on Add to Playlist button, the function below is fired.

As you can see that addVideoToPlaylist() function is reused here (already been discussed).

/\*

\* Adds Featured Videos to Playlist

\*/

addFeaturedVideoToPlaylist: function(button) {

var me = this,

selectedVideo = me.getSelectedVideo(button);

me.addVideoToPlaylist(selectedVideo, false);

},

Here, we needed to get hold of the selected item data of the list. To do that we wrote the getSeclectedVideo() function. In this function we used the up() method to get the listContainer and caught the videoList in action. Having these two references, our task becomes simple where we just need to get the selected item data using getSelection() and getData() in conjunction. Now, this selectedVideo is returned to the event function above which an addVideoToPlaylist() is called, using this data as parameter.

/\*

\* Get Selected Video Item

\*/

getSelectedVideo: function(button) {

var me = this,

listContainer = button.up().up(),

videoList = listContainer.getItems().items[1],

selectedVideo;

selectedVideo = videoList.getSelection()[0].getData();

selectedVideo.vid = selectedVideo.id;

return selectedVideo;

},

### Persisting Playlist with the help of Local Storage

localStorage is extremely useful for saving user-specific information without the need of building server-side infrastructure, for supporting it. Let's imagine we're creating a playlist of videos and want to save the playlist data locally so it can be easily accessible again later. We'll start by creating a playlist store which requires Video (Video.js) model:

Ext.define('BoomBoom.store.PlaylistVideos', {

extend: 'Ext.data.Store',

requires: ['Ext.data.proxy.LocalStorage'],

config: {

model: 'BoomBoom.model.Video',

autoLoad: true,

identifier: 'uuid',

proxy: {

type: 'localstorage',

id: 'playlistlocalstorage'

}

}

});

Our playlist store contains autoLoad, identifier and a Proxy definition. As autoLoad is true, the load() method is automatically called after creation. The load() method also adds the localstorage data to our playlist store whenever our app is accessed. As identifier is 'uuid', our identifier generation strategy for the playlist store ensures unique id. The only configuration we need to pass to the LocalStorage proxy is an [id](http://localhost/sencha/touch-docs-2.3.0/#!/api/Ext.data.proxy.LocalStorage-cfg-id). This is important as it separates the store data, in this Proxy, from all others. The localStorage API puts all data into a single shared namespace, so by setting an id we enable LocalStorageProxy to manage the saved playlist data.

Saving our data into localStorage is easy and usually done with a [Store](http://localhost/sencha/touch-docs-2.3.0/#!/api/Ext.data.Store):

//Creating videoInfo object which will contain all the information of a video

videoInfo = {

title: recordData.title,

channelTitle: recordData.channelTitle || 'Annonymous',

id: recordData.id,

vid: recordData.videoId,

thumbnail: recordData.thumbnail,

duration: recordData.duration

};

//Adding videoInfo object to playlist store

var playlistStore = Ext.getStore('PlaylistVideos');

playlistStore.add(videoInfo);

//finally, save our playlist store data to localStorage using inbuilt function sync()

playlistStore.sync();

|  |
| --- |
| **Limitations**  If the proxy is used in a browser where local storage is not supported, the constructor will throw an error. A local storage proxy requires a unique ID which is used as a key that stores all record data in the local storage object.  It's important to supply this unique ID as it cannot be reliably determined otherwise. If no id is provided but the attached store has a storeId, the storeId will be used. If neither option is presented the proxy will throw an error. |

## Playing videos from different views

We have discussed how to search and add videos to playlist. Now is the time to know how to play them. We can play videos in two ways:

1) By clicking on the play button, that appears when you click on any videos from categorised lists in the home view or from the searchlist view.

2) You can first add video to playlist and then tap on the video item in the playlist.

|  |  |
| --- | --- |
| searchlist.png | playlisttiitanic.png |

The first option is a direct way to play a video. Let’s see what exactly happens when you click the play button. Upon click, two actions are performed, the first is that the video item is added to playlist and then the video is played.

The second approach is rather simple and performs only one action - playing the video.

By now, you already know the background code for adding videos to playlist so let us see how to play them. First we will look at the code to play video using the first way i.e, from the buttons that appear in the home page and search page (in App.js Controller).

/\*

\* Play selected video

\*/

playVideo: function(recordData) {

var me = this,

videoPlayer = me.getVideoContentPage(),

menuIndex = me.util.getMenuIndex("videoplayer", me),

count, index;

me.redirectTo('videocontentpage');

me.getVideoPlayerContainer().setMasked({

xtype: 'loadmask'

});

index = me.addVideoToPlaylist(recordData, true);

me.getMenu().select(menuIndex);

if (me.getVideoPlayerContainer()) {

me.getVideoPlayerContainer().setData(recordData);

}

Ext.defer(function() {

count = Ext.getStore('PlaylistVideos').getCount();

me.getVideoPlayerContainer().setMasked(false);

if (!(index >= 0)) {

index = count - 1;

}

me.getPlayList().select(index);

me.util.disablePlayerControls(me, index, count);

}, 1000);

}

Some of the methods used here are not yet discussed but we will definitely check them in greater detail, later. For now you just need to know their function, by reading the comments written above them.

Here, we are skipping the refs and control part, intending to discuss only the function which is invoked under the tap event of the play button.

<script type="text/template" id="video\_player\_template">

<tpl if="vid">

<div class="video-player"><iframe type='text/html' src='https://www.youtube.com/embed/{vid}' width='100%' frameborder='0' fullscreen=true/>

</div>

<tpl else>

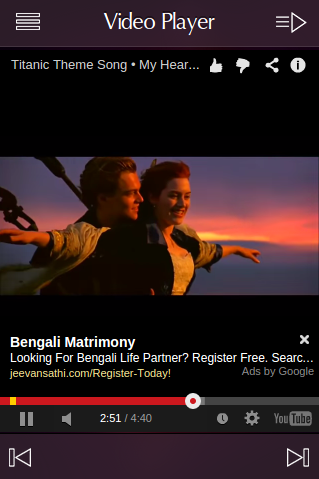
<div class="video-player-empty-text">No video to Play</div>

</tpl>

</script>

For more details about the video player container that holds the template, you may refer to the ‘VideoPlayer’ view.

As we observe the above template we can see the {vid} value is passed on to the template when the data is set for our video container. This ‘vid’ completes the link and the HTML5 player takes care of the rest i.e, your video is up and running.



The second way is much more simpler and is implemented using small function in Player.js Controller. Let’s have a look:

/\*

\* Plays the selected video

\*/

playVideo: function(index) {

var me = this,

videoPlayer = me.getVideoContentPage(),

// Getting the selected item data from playlist videos store

videoId = Ext.getStore('PlaylistVideos').getAt(index).data,

index = me.util.getMenuIndex(‘videoplayer’, me);

// Setting the data to the video container’s template

me.getVideoPlayerContainer().setData(videoId);

me.util.showActiveItem(me, videoPlayer, 'slide', 'left');

me.getMenu().select(index);

me.getPlayPrev().setDisabled(false);

me.getPlayNext().setDisabled(false);

me.util.disablePlayerControls(me, videoIndex, videoCount);

},

The only difference between both the functions is the origin of the data source. Here, we are getting data from the ‘PlaylistVideos’ store where index corresponds to the index of the selected/tapped item on the video playlist, that the user wishes to watch. The rest is pretty much the same as before.

Playing video from Home view is quite similar to adding video to playlist, from home view, with a small difference which is quite self explanatory on its own..

/\*

\* Plays Featured Video

\*/

playFeaturedVideo: function(button) {

var me = this,

selectedVideo = me.getSelectedVideo(button);

me.playVideo(selectedVideo);

},

## Removing videos from playlist just by a swipe

Till now we have searched videos and added them to the playlist. The Playlist contains the list of videos of our choice and lets us rewatch watch them whenever we want. What if you no longer want the video to be there in the playlist? Or perhaps you have added a video to the playlist by mistake? The solution is simple, remove the video from playlist. Yes, we have added this feature to our app. You can remove any video from the playlist at any point of time just by swiping. When you swipe an item in the playlist, there appears a delete icon. Now tap on that icon and you are done with removing the video. It is that simple. There is nothing to worry if you have tapped the delete icon by mistake. We have put a confirmation message there which awaits your permission. When you tap on ‘yes’ the video will be removed.

If your playlist has become over crowded and you want to clear all the videos, there is an option. We have kept a ‘Clear All’ button at the bottom of the playlist, tapping which removes all the videos from the playlist.

|  |  |
| --- | --- |
| playlistm.png | deletes.png |

Let’s have a quick look at the functions

/\*

\* Delete icon is displayed when swipped an playlist item

\*/

showDeleteIcon: function(view, index, target, record, event) {

var swippedListItem = Ext.get(target.element);

if (event.direction === "left") {

// moves the item to left to reveal delete icon

swippedListItem.removeCls('in').addCls('out');

}

if (event.direction == "right") {

// moves the item to its original position

swippedListItem.removeCls('out').addCls('in');

}

}

showDeleteIcon() function reveals the delete icon when you swipe the item in playlist. Initially the icon is kept hidden and appears only when an item is swiped to the left. In case you don’t want to delete the video, swipe the item in opposite direction. The delete icon will disappear again and the item takes back to its original position. We have used a simple CSS design to implement this trick.

/\*

\* Selected item is deleted or played

\*/

deleteVideoFromPlaylist: function(view, index, target, record, event) {

var me = this,

clsArray = [],

tappedListItem = Ext.get(target.element);

if (event.target.className === 'playlist-delete-icon-wrapper' ||

event.target.className === 'delete-icon') {

for (var i = 0; i < Ext.getStore('PlaylistVideos').getCount(); i++) {

clsArray[i] = me.getVideoPlaylist().getItemAt(i).element.hasCls('out');

}

try {

Ext.Msg.confirm("Confirmation", lang.confirmMsg.playlist, function(buttonId) {

if (buttonId === 'yes') {

var playlistStore = Ext.getStore('PlaylistVideos');

playlistStore.removeAt(index);

playlistStore.sync();

if (!Ext.getStore('PlaylistVideos').getCount()) {

me.getPlaylistClearButton().setDisabled(true);

}

for (var i = index + 1; i < clsArray.length; i++) {

if (clsArray[i] === true) {

me.getVideoPlaylist().getItemAt(i - 1).element.addCls('out');

} else {

me.getVideoPlaylist().getItemAt(i - 1).element.removeCls('out');

}

}

//Disabling 'Clear All' button if playlist is empty

if (!Ext.getStore('PlaylistVideos').getCount()) {

me.getPlaylistClearButton().setDisabled(true);

me.getPlayPrev().setDisabled(true);

me.getPlayNext().setDisabled(true);

}

}

});

} catch (error) {

console.log(error);

}

return;

} else {

tappedListItem.removeCls('out').addCls('in');

me.togglePlaylist();

//plays an item

me.playVideo(index);

}

},

What does the function deleteVideoFromPlaylist do for us? It removes videos from playlist. If you have agreed to remove the video by tapping the yes option displayed in the message box, the video gets removed from the playlist and also from the store. When the playlist becomes empty the button at the bottom is disabled.

Take a look at these lines of code:

for (var i = index + 1; i < clsArray.length; i++) {

if (clsArray[i] === true) {

me.getVideoPlaylist().getItemAt(i - 1).element.addCls('out');

} else {

me.getVideoPlaylist().getItemAt(i - 1).element.removeCls('out');

}

}

When a list item is deleted from list, sencha by default holds the previous state of that list index ,i.e if before delete, the list item had class ‘out’ to it then after delete, the immediate next list item when taking its position will also have class ‘out’ to it.This was an undesired effect for us so we wrote a fix shown above which is left for you to decode.

/\*

\* Clear all items from playlist

\*/

clearPlaylist: function() {

var me = this,

videoPlaylistStore = Ext.getStore('PlaylistVideos');

Ext.Msg.confirm("Confirmation", lang.confirmMsg.clearPlaylist, function(buttonId) {

if (buttonId === 'yes') {

videoPlaylistStore.removeAll();

videoPlaylistStore.sync();

me.getPlaylistClearButton().setDisabled(true);

}

});

}

clearPlaylist function simply removes all the videos from the playlist and the store when the ‘Clear All’ button is tapped.

|  |  |
| --- | --- |
| playlistm.png | deleteall.png |

**Customized video player controls**

As the name suggests this page works as the container for playing videos. Apart from playing the video we have added few more functionalities to this page to make it more user friendly. From the Video Player page the user can easily open the navigation menu and go to any page of the app, or he can directly go to the playlist page. This can be done by simply clicking the buttons on the top left and top right corner of the video player page respectively.

At the bottom of the page we have added two more buttons, one at the left corner and another at the right corner to go to the previous or next option as saved in the playlist. This will let users watch the videos one after another without going to the playlist page repeatedly.



Moving from one card to another is rather an easy task, than playing one video after another using next or previous button. Here is an easy approach to solve that. Take a look at the functions used in Player.js Controller. Before looking into them don’t forget to check the refs and control required for the said functionality,

refs:{

...

// Views

videoPlaylist: 'list[name="video\_playlist"]',

videoPlayer: 'videoplayer',

videoContentPage: 'videocontentpage',

// Container

videoPlayerContainer: 'container[name="video\_player\_container"]',

// Buttons

playPrev: 'button[name="play\_prev"]',

playNext: 'button[name="play\_next"]',

...

},

control:{

playPrev: {

tap: 'playPrevVideo'

},

playNext: {

tap: 'playNextVideo'

}

}

Now, we will focus on next/ previous feature. For that, we will discuss 3 functions in detail.

By looking at the refs and control we already know that when ‘playPrev’ button is tapped ‘playPrevVideo()’ method is called and when ‘playNext’ button is tapped ‘playNextVideo()’ method is called. Now, before understanding these two functions you need to know another function called ‘getSelectedIndex()’.

/\*

\* Getting Index of the selected playlist item

\*/

getSelectedIndex: function() {

var me = this,

videoPlaylist = me.getVideoPlaylist(),

// selectedItem is an array of selected items

selectedItem = videoPlaylist.getSelection(),

videoPlaylistStore = videoPlaylist.getStore(),

videoPlaylistStoreCount = videoPlaylistStore.getCount(),

index = "";

// checked whether selectedItem array has a length greater than 0

if (selectedItem.length > 0) {

index = videoPlaylistStore.indexOf(selectedItem[0]);

}

return {

'index': index,

'count': videoPlaylistStoreCount,

'list': videoPlaylist

};

},

In this function, we retrieve the index of the video currently selected or being played in the lists Store. First we have referenced the video playlist, followed by its store and found out the current item selected in the video playlist. We have also counted the items in the videoplaylist’s store. Now, after the check, we get the index of the selected item of the video playlist by using the indexOf() method on the video playlist store’s instance. Then we return an object with the index, the count and the video playlist’s instance.

Once we know the index of current selected item in video playlist store, we just have to increase the index when user taps next button or decrease the index when user taps previous button.

playPrevVideo: function() {

var me = this,

selectedItem = me.getSelectedIndex(),

index = selectedItem.index,

count = selectedItem.count,

list = selectedItem.list;

index--;

if (index >= 0) {

list.select(index);

me.playVideo(index);

}

},

/\*

\* Plays the next video in the playlist

\*/

playNextVideo: function() {

var me = this,

selectedItem = me.getSelectedIndex(),

index = selectedItem.index,

count = selectedItem.count,

list = selectedItem.list;

index++;

if (index < count) {

list.select(index);

me.playVideo(index);

}

}

After altering the index based on the invoked function, we will dynamically select the item on the list with the altered index, equivalent to its index and then we will invoke the play video function with the parameter set to the altered index.

## Routing in BoomBoom

Two typical shortcomings of single-page Mobile apps are incorrect behavior of device’s “back” button, and the inability to link directly to deep content. Thankfully, the Sencha Touch 2 MVC framework includes powerful support for [history tracking and deep linking](http://docs.sencha.com/touch/2-0/#!/guide/history_support), commonly referred to as ‘[routes](http://docs.sencha.com/touch/2-0/#!/api/Ext.app.Controller-cfg-routes)’.

As of Sencha Touch 2, Controllers can directly specify which routes they are interested in. This enables us to provide history support within our app, as well as the ability to deeply link to any part of the application that we provide a route for.

The routes map the contents of the browser address bar to a Controller function that is called when the route is matched. The routes can be simple text like the login route, which matches against http://myapp.com/#login, or may contain wildcards such as the 'user/:id' route, which matches urls like http://myapp.com/#user/123. Whenever the address changes, the Controller automatically calls the specified function.

We have used three routes corresponding to three views inside controller(App.js):

routes: {

'home': 'gotoHome',

'searchList': 'gotoSearchList',

'videocontentpage': 'gotoVideoContentPage'

}

The left hand side denotes route name and the right hand side denotes the function to be called, when the route is directed to. For example, when ‘home’ route is directed to, ‘gotoHome()’ function will be called and this function will show the ‘Home’ page on the viewport. The implementation of ‘gotoHome()’ function is given below:

gotoHome: function() {

var me = this;

me.util.showActiveItem(me, me.getHome(), 'slide', 'right');

}

‘showActiveItem()’ function is written inside util file and it is as follows:

showActiveItem: function(me, view, slide, direction) {

//getMenuContent() returns the ‘content’ view containing ‘Home’,’VideoPlayer’ and ‘SearchList’ views using

card layout

me.getMenuContent().animateActiveItem(view, {

type: slide || '',

direction: direction || 'left'

});

},

To show ‘Home’ page, we just write **this.redirectTo(‘home’)** instead of using gotoHome() function.

Here ‘this’ refers to the controller inside which the route is defined.

In the function navigatePage(), we have effectively used redirectTo() function to display different views in the viewport via card layout.

//This function is called whenever any item of menu list is tapped

navigatePage: function(list, index, target, record) {

//Used for sliding purpose of menu list

this.toggleNav();

switch (record.get('title')) {

case "Home":

me.redirectTo('home');

break;

case "Search":

me.redirectTo('searchList');

break;

case "Video Player":

me.redirectTo('videocontentpage');

break;

}

}

## Some technicalities to enhance our app

### Slide Menu

As stated earlier, Main.js is the parent container which holds the Menu View(Menu.js) and the Content View(Content.js).

Let’s take a look of the Main View:

Ext.define('BoomBoom.view.Main', {

extend: 'Ext.Container',

xtype: 'main',

config: {

layout: 'hbox',

items: [{

xtype: 'content',

cls: 'slide',

width: '100%'

}, {

xtype: 'menu',

width: 150

}]

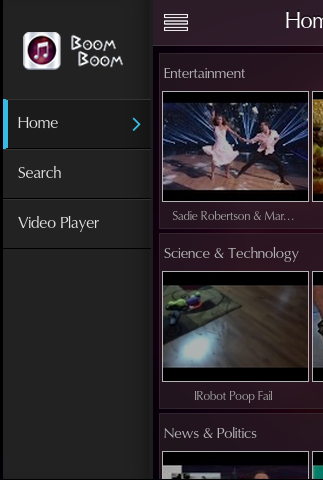
}

});

The layout of the Main view is very important as it sets the base for the application. We have set the layout of the page to ‘hbox’ with two items: 1) menu with a custom width of 150px, 2) content which takes the full width initially but on tapping the menu icon in toolbar the menu slide in from left to a width of 150px appearing as a side menu. We have used simple CSS to bring in the slide effect of the Menu View.

For a detailed explanation with more examples you can see [Slide navigation with Sencha Touch](http://innofied.com/simplest-slide-navigation-with-sencha-touch-2-2/)

Now have a look at our Navigation View.



The functions to bring the sliding effect of the Menu is implemented in App.js Controller. First we have referenced the navigation button on the toolbar as ‘navBtn’ and under tap event in control part we have written the sliding menu function ‘toggleNav’.Once we click on an item in the menu list, the same function is again fired with an objective to close the menu and navigate to the selected item page.

### Sliding playlist attached to the video player

This mechanism is quite similar to the Slide Menu discussed above and so we are going to let you crack the solution!!!

### Featured list and Custom Dataview

Lets begin with the design that we had in mind for the Home Page. Below are the designs of the Home page for our app.

Before getting on with the details of the Home Page let’s continue on our trail with the Content Page a little more. The content page has a fit layout, containing an item with card layout for holding multiple views specifically ‘Home’, ‘Search’, ‘VideoPlayer’ and ‘Playlist views’.

Now, let’s come back to the Home Page. It has categories as horizontal scrollable lists. Without creating different views for each lists we can always optimize our code by creating one view ‘FeaturedList’ and using it multiple times by creating multiple instances of it.

If you notice closely, whenever there are multiple scrolls in a single page/view, scrolling effect is undermined. For example, in our app we have four horizontal scrolling lists and overall vertical scroll for the entire page. So, it is common that when you are scrolling one list (horizontal) it will also affect vertical scroll because of the undirected touch pressure of our fingers. Overall, it looks odd and feels bad. To avoid this undesired effect we have written a ‘custom dataview’ view. Let’s take a look at ‘CustomDataview’ code.

Ext.define('BoomBoom.view.CustomDataview', {

extend: 'Ext.dataview.DataView',

xtype: 'customdataview',

parentPanel: null, // Adding parentPanel to config of 'customdataview'

config: {

listeners: {

painted: function() {

var me = this,

parentPanel = me.parentPanel || me.config.parentPanel;

me.getScrollable().getScroller().on('scrollstart', function() {

parentPanel.setScrollable(false);

}, me);

me.getScrollable().getScroller().on('scrollend', function() {

parentPanel.setScrollable(true);

}, me);

}

}

}

});

Now we will take a look at the ‘FeaturedList’ view.

Ext.define('BoomBoom.view.FeaturedList', {

// Custom Dataview features are inherited

extend: 'BoomBoom.view.CustomDataview',

xtype: 'featuredlist',

config: {

height: 100,

inline: {

wrap: false

},

scrollable: {

direction: 'horizontal'

},

itemCls:'featured-list-items',

itemTpl:'<div class="album-art"><img src="{thumbnail}"></img></div>',

}

});

In the above code we have extended CustomDataview as it holds our secret to solve our scrolling problem. For detailed explanation and more examples you can check, [Single scrolled parent with multiple containers in Sencha](http://innofied.com/single-scrolled-parent-with-multiple-containers-in-sencha/).

### Activating navigation menu list according to the card in view

It becomes quite a problem when you navigate from one page to another without using navigation menu. What happens is that corresponding menu item is not selected as per the content displayed. For example, in this app you can navigate from videoplayer to playlist and vice-versa. But when we navigate from playlist to video player and then open the slide navigation menu, we find that playlist is active in menu instead of Video Player, which is unacceptable. So we came up with a small function to effectively solves this issue. Here’s the function in Util.js:

/\*

\* Get index of requested page

\*/

getMenuIndex: function(action,me){

var menuStore=me.getMenu().getStore(),

menuStoreData = menuStore.getData(),

count = menuStore.getCount();

for(var i=0;i<count;i++){

if(menuStoreData.items[i].data.action === action){

return i;

}

}

}

Time to discuss how this function works. At first you need to know that in the data contents of Menu list we have included an action field which acts as a unique identifier, of the item in the list, helping us find the index of that item in its store. To do all this, we first get the store data of the menu list and count the items in the store. Then we use a for loop to check for a match with the action field value we passed to the item ‘action value’. On successful match, the function returns the index. After that we select the item in menu list having the specific index, as shown in the next two lines of code.

index = me.util.getMenuIndex("videoplayer", me);

me.getMenu().select(index);

### Utility file

For every application, we need a set of common functions and properties that will be used throughout the application. We created a Util singleton class for the same thing and put the file under the app/util/ directory. You do not need to understand the functions of this file right here. We'll keep discussing these functions as we move forward.

Ext.define('BoomBoom.util.Util', {

requires: ['Ext.MessageBox'],

singleton: true,

api: (function() {

var baseUrl = 'https://www.googleapis.com/youtube/v3/';

// var baseUrl = 'php/action.php?https://www.googleapis.com/youtube/v3/';

return {

videoUrl: baseUrl + 'videos',

searchUrl: baseUrl + 'search',

key: 'AIzaSyD6FvoLaIFqyQGoEY4oV7TEWGAJSlDd1-8'

}

})(),

failedRequest: function(errorText) {

var error = errorText || 'Connection Error';

Ext.Msg.alert('Error', error, Ext.emptyFn);

},

/\*

\* Get index of requested page

\*/

getMenuIndex: function(action, me) {

var menuStore = me.getMenu().getStore(),

menuStoreData = menuStore.getData(),

count = menuStore.getCount();

for (var i = 0; i < count; i++) {

if (menuStoreData.items[i].data.action === action) {

return i;

}

}

},

/\*

\* Shows the Active Item (With Slide)

\*/

//slide the view into a particular direction using built in function animateActiveItem()

showActiveItem: function(me, view, slide, direction) {

me.getMenuContent().animateActiveItem(view, {

type: slide || '',

direction: direction || 'left'

});

}

/\*

\* Disables the player next previous controls

\*/

disablePlayerControls: function(me, videoIndex, videoCount) {

if (videoIndex === videoCount - 1) {

me.getPlayNext().setDisabled(true);

}

if (videoIndex === 0) {

me.getPlayPrev().setDisabled(true);

};

}

});

### Language file

In our application there are messages that are displayed under different situations. Instead of writing these messages directly in the functions, we create a lang object for the message and put the file in the directory Lang/lang.js. This is beneficial when we are using same set of messages in different places. In this app we have not used the same set of messages but it is a good practise to keep the messages inside a lang file. It is actually used for internationalization and can be beneficial in creating a multi language app. Just creating a lang.js file is not sufficient. We import the lang.js file inside the index.html via script tag to get access to the lang object.

The script tag inside index.html file looks like:

<**script** src="lang/Lang.js" type="text/javascript"></**script**>

Have a look how we use the lang object in our functions:

Ext.Msg.confirm("Confirmation", lang.confirmMsg.clearPlaylist, function(buttonId) {

if (buttonId == 'yes') {

videoPlaylistStore.removeAll();

me.util.refreshLocalStorage();

me.getPlaylistClearButton().setDisabled(true);

}

});

}

And this is how the lang.js file looks like:

lang = {

confirmMsg: {

playlist: "Are you sure you want to delete this video from playlist?",

clearPlaylist: "Are you sure you want to delete all videos from the playlist?"

}

}

# Some thoughts for next version

Development is a field where there is always a scope for improvement. We have achieved our initial objectives with this version of BoomBoom. But we have some more tricks up our sleeves to integrate with this app and improve its functionalities. Some of them are:

* Showing a suggestion list when users search videos
* Displaying a larger search result
* Integrating music player into the app
* Implementing user module with individual accounts

When we come up with our new version we will certainly update this tutorial. Till then, Happy Booming!!!

# 