## Design Decisions:

### Single Page Architecture:

I chose single page architecture since it provides a very elegant user experience without reloading the page. Also it makes the application faster and efficient as it runs on the client side by loading the scripts on demand. This application involved a lot of user interaction with changing artists and playing videos, so reloading the page on each time will be slower and less user friendly.

### Building with ES6 , Babel.js and Browserify

I decided to use latest JavaScript technologies such as ES6 , babel.js and browserify in order to speed up the development process and leverage the benefit of node.js modules. I implemented a small Gulp.js script to transpile and browserify the node.js code and bundle into browser compatible JavaScript.

To improve the page loading speed on client side Gulp script creates a build folder with all the transpiled files and bundles them into ‘bundle.js’.CSS files of all the components are also concatenated into single ‘style.css’ file that is loaded on client side.

### Modular Design with React and Flux:

I decided to use Facebook’s React library for two reasons: First, It provides in-built support for server-side rendering and secondly because of its modular component based architecture that allows you to build reusable and highly maintainable components. I decided to use a simple Flux application architecture that complements React.js’s unidirectional data flow and manage data stores efficiently.

Each component is a separate module file in components directory that implements individual react component. Each class can import and render other components as and when required. All the modules are encapsulated in “PlaylistContainer” component that acts as a parent for routing the events and unidirectional data flow.

### Responsive Design:

The entire application is designed to be responsive and works very well with small as well as large screens. Every component that I have developed is responsive in nature. Each component has its own CSS file containing queries to resize and rearranges itself based on the users screen size. This makes it very mobile friendly.

### Youtube Player Component:

I tried out existing node module ‘React-Youtube’ for implementing youtube player, but it caused page to reload every time I played a new video, which was very undesirable for a Single page application. So I implemented my own React component for youtube player that loads the youtube API asynchronously and allows you to load the videos and playlists without reloading the page using youtube’s API.

I am using Youtube’s latest Iframe API to implement the video player. This API decides at runtime what kind of videoplayer to display (HTML5 or flash or SWF) based on client’s browser compatibility. Hence it was not possible to render the player on server side.