**Online Course Player with Socket.IO**

This online course player is implemented with Socket.IO and Node.js. Another implementation with the same functionalities is developed by SignalR and ASP.NET, you can read the posting about it from my portfolio. Both of the implementations are based on WebSocket. For the basic knowledge of WebSocket and Socket.IO, please refer to my blog posting Develop Realtime Online Application with WebSocket.

1. **Introduction**

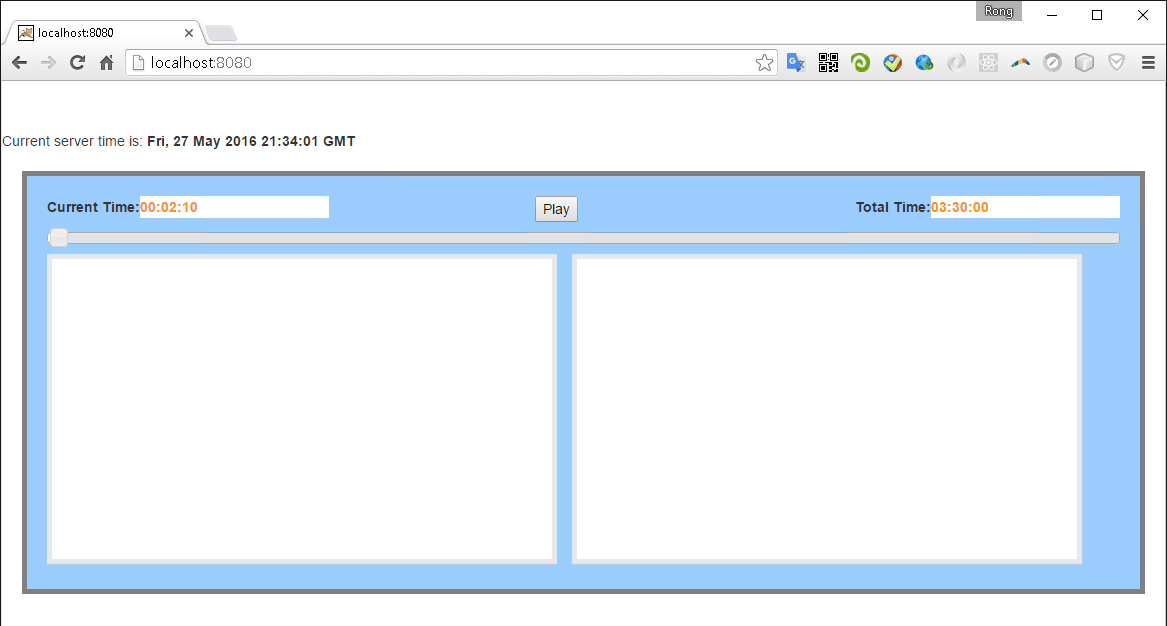
A course player consists of three components, video, screenshot and whiteboard.

* Video is captured by a camera during the lecturing time, and saved as mp4.
* Screenshot is captured from computer monitor through which teachers share their handouts/materials to the students. Screenshot are images which are saved in a single file.
* Whiteboard is captured from special pens and boards. Any operation on the board, such as writing, drawing or brushing is logged and stored to a single file.

1. **UI**

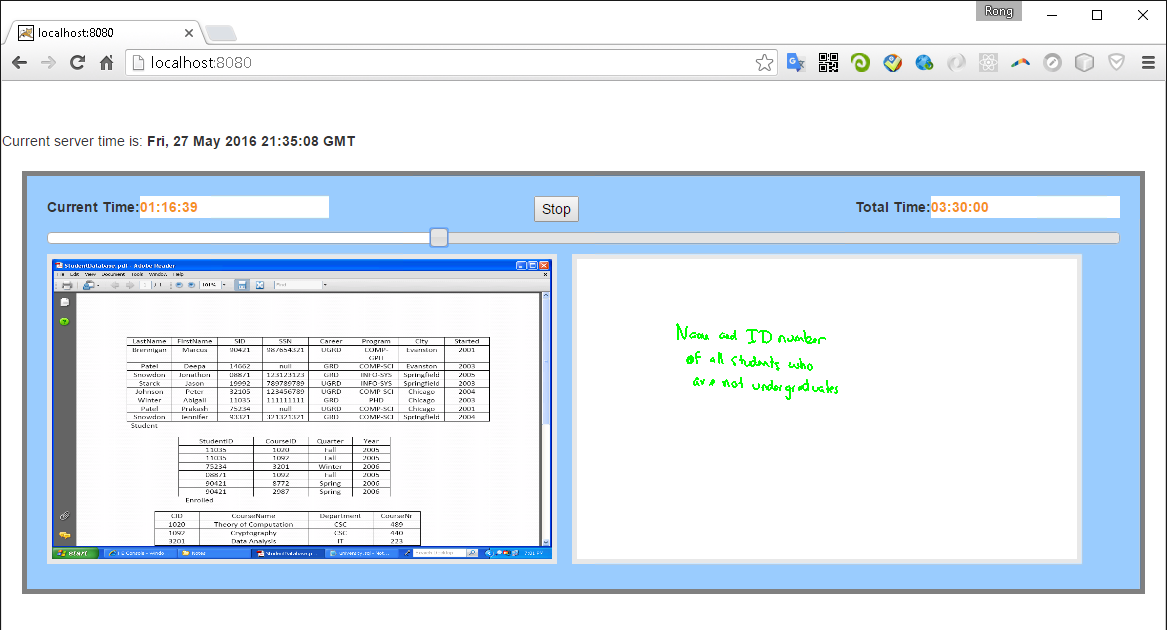
For this course player, video is played independently. The content of the screenshot and whiteboard is synchronized with the playing process of the video. In this sample, I use a slider bar to simulate the video player.

On the top of the player, there is the process bar and a Play button. There are two canvases below the process bar. The left one is for screenshot and the right one is for whiteboard.



1. **Play**

When you click the play button, the process begins to move, the current time will be refreshed as well, one second for interval. The screenshot and whiteboard canvas show the content simultaneously. You can drag the process bar forward or afterward.



1. **Under the Hood**

How does this dummy player work?

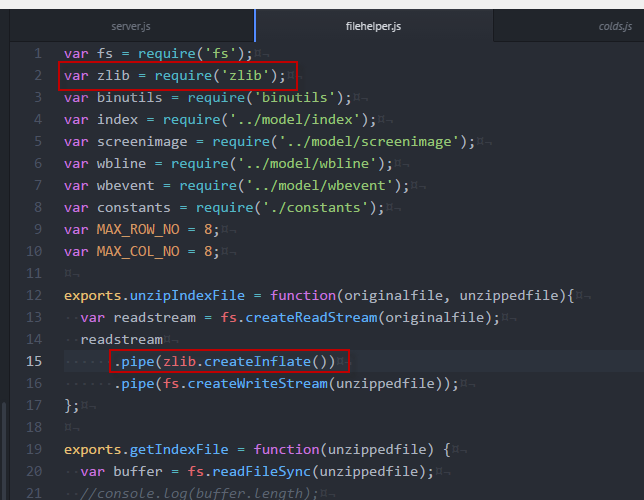
1. When the page is opened, the connection is setup between server and client(web browser).
2. Next, the course starts to play after the Play button is clicked. At the same time, a notification is sent to the server, and the server knows that the player has begun to work.
3. Then, server starts a timer, run the task(step 4) for every second.
4. Server reads data for screenshot and whiteboard based on the current time.
5. If there is any update(new image or new drawing), it will send data(JSON format) to client. Otherwise, no communication occurs from server to client.
6. If web browser gets data, it will draw images for screenshot or lines for whiteboard accordingly.
7. The communication from client to server occurs only when the play button is clicked or the process bar is dragged.
8. The communication from server to client occurs only when new data is found.
9. **Codes**
   1. Http Server and Socket.IO(Server side)

Use express to setup an http server. And use socket.io to transfer data between client and server. There are two broadcasting event, ‘draw’ for screenshot and ‘drawline’ for whiteboard.



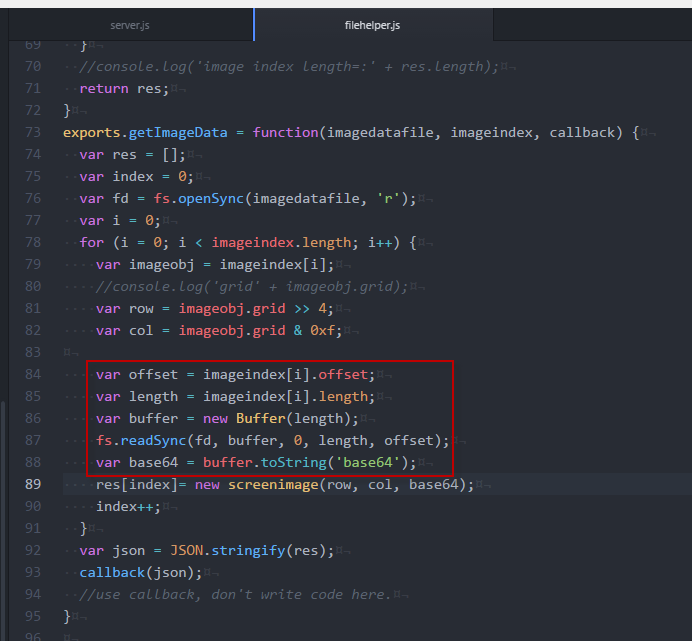
* 1. Unzip file (server side)

The data files for screenshot and whiteboard are compressed. We use the zlib module which is also provided by Node.js to decompress the file. Generally, there are two method for zip or unzip, Gzip and Inflate. Here, we use the ‘Inflate’ method. For your project, you must choose the right one for the compressed file.



* 1. Read Data File (server side)

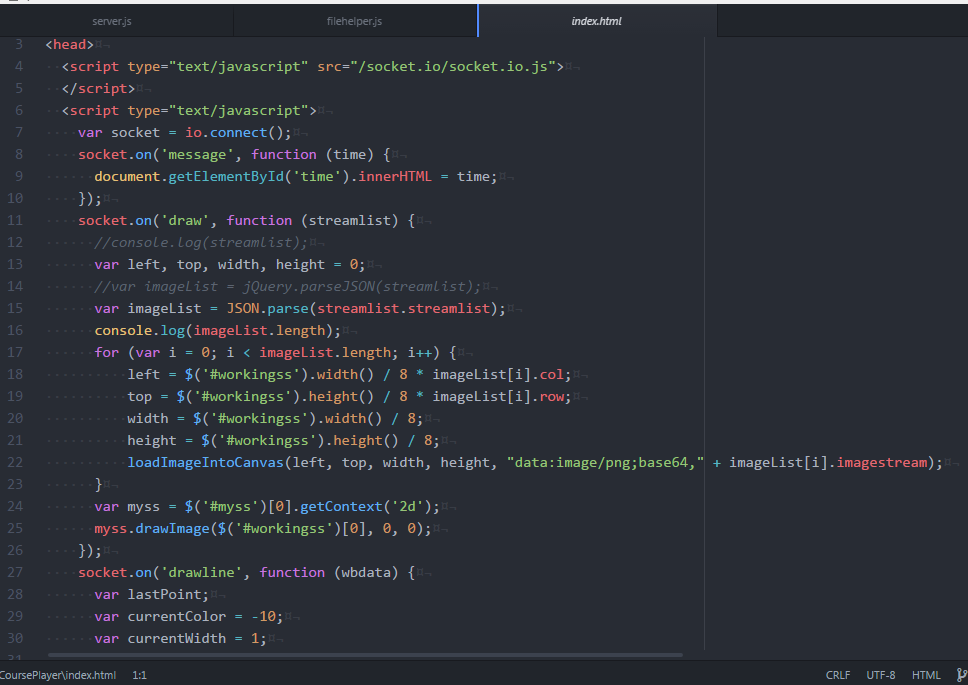
Use files system and Buffer provided by Node.js to read data from files. Notice that, we read data from stream, offset and length must be specified. For each screenshot, it consists of 8\*8=64 pieces of images. For each image, we use base64 format and draw it to canvas later. All the images are converted to JSON string format and sent to client.



* 1. Draw Images

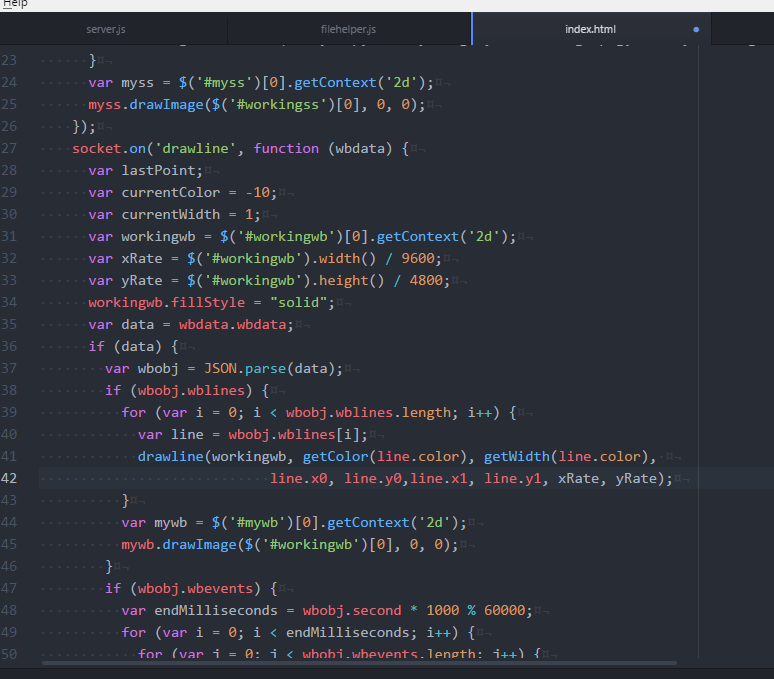
On the client side, we monitor the ‘draw’ event and receive data from server side.

For each screenshot, there are 64 images for maximum. There will be fewer if some parts of the screenshot are not changed. First, we need to parse the data to JSON format. Then we draw the images one by one to a hidden canvas(#workingss). After all are finished, draw the whole hidden canvas to the visible screenshot canvas(#myss).



* 1. Draw Lines

Similar with images, we need to parse the data to JSON format first. Then, draw lines according to the color, width, and position.



1. **Conclusion**
   1. Easy to Implement

If you are familiar with Node.js and javascript, it is not too difficult to develop such real time online application.

* 1. Low Bandwidth Consumption

Communication occurs only when necessary. Unlike traditional web application, WebSocket makes the web application react at real time. This improve the user experience at client side and system performance at server side.

* 1. Cross-platform

This player is web based, no installation on client’s machine is required. Besides, this course player is based on HTML5, so it can be accessed in different web browsers and on different platforms. No need to install extra plugin in web browser, such as flash player or Silverlight.