Interactive Canvas

A demo of the exciting features of HTML5 canvas and WebSocket

GUO Yuxiang
WU Sisi
ZHANG Yaofeng
ZHANG Yusi
ZHAO Guanlun

Abstract

The Interactive Painter is an online application that allows multiple users to interactively paint on one canvas and communicate with each other at the same time. Based on the web socket mechanism, this application works quite efficiently and can deal with relatively large amount of data transfer. The design and development of this web application is motivated by Google Docs, which is a convenient platform for collaborations. Since a graphical version is not available, we started this project to design and implement this painter.

1 Motivation

The motivation of this project is that there lacks a user-friendly online cooperation drawing platform.

A online interactive canvas can make it much easier for people having online meetings or discussion by allowing the users to draw on a piece of canvas and convey their ideas more effectively and comfortably.

However, now we don't have a relatively well-developed application available like this. If someone, for example, a designer, wishes to describe his or her design to someone else online, he or she will most probably choose to send emails and this is clearly not as convenient as having an interactive platform to draw on while keep others updated simultaneously.

What's more, if some organization needs to have an online meeting in which the members are to discuss a plan, a interactive picture can usually provide a much more straightforward medium for the exchange

of opinions.

Thus we thought of implementing an interactive canvas to make life easier. Hopefully we will continue with this project and develop it into a full-functional web application with more friendly user interface as well as more advanced features.

2 Basic Idea

Here we briefly explains how the application works from the perspective of a user.

When the user enters the webpage of the our interactive canvas, he or she will firstly be promoted to enter a username. After that the main interface is entered and the user can now start to draw on the canvas or chat with other users.

When the user draws on the canvas or send messages through the chatting module, a message, stringified with JSON, with be sent to the server and the server, according to different circumstances, will use the database and send another message back to the user or all the users. Then the view of the webpage is changed. This is basically one single step of the data communication.

More complicated data communication will be covered in the following sections.

When the user logs out, the user will be deleted from the server but the data will be kept in the server's database.

3 Design

The design of this application is mainly about the client side canvas mechanism and server-side database, also including the data transfer and user interface.

- *Canvas Mechanism*: The structure of the canvases is quite complicated in this application. The main reason for this is the implementation of undo and redo functionalities. This will be covered later in the "highlighted features" section. Here we mainly introduce the use of the canvas to display the line segments: When the client side **onmessage()** function is triggered by a message containing a drawing request from the server, the function **draw_canvas()** is called, drawing on the canvas according to what is specified in the message. This works for both the user who had drawn this line segment and other users. The canvas

also works in catching the **mouseevent**s generated by the user's mouseclick, mousemove and so on. When a valid mouseevent is caught a message is send to the server. Thus the canvas is the central part of the client side implementation.

- *Database*: The database mainly consists of two parts: one for the line segments and the other for chatting history. The database for chatting history is quite simple, which is no more than a normal array. The database for line segments are more complex, which consists of a static database and some buffers, whose number is equal to the number of current users. The database only store the segments that are not to be changed and buffers are for the segments that can be "undoed" and "redoed".

- *Data Transfer*: The data transfer part is supported by *JSON*, the data-interchange format. The types of data transfers in this application is quite varied, which can be categorized into the following parts: user login, user logout, drawing segments, starting and ending segments, undo, redo, chatting and so on. All the requests have different labels that make them easily recognizable by the server and the client-side programs.

- *User Interface*: The user interface design of the interactive painter is inspired by the UI of Google Docs, which uses a quite refreshing theme with simple lines and only a few kinds of color. We chose it because our application is aimed to be simple and elegant, and this style is quite suitable.

4 Highlighted Features

Our application also includes some highlighted features that make it more user-friendly and more elegant.

4.1 Multiple Canvas Mechanism

The most interesting and complicated part of this application is the implementation of the multiple canvas mechanism and it is adapted for several reasons.

Firstly, it would be very inconvenient if a painter does not have undo and redo functions. The implementation of these functions is so trivial on local, non-interactive canvas applications, that what needs to be done is merely saving the current canvas each time the mouse is pressed down and save them in a stack. However in an interactive application this approach is not as practical. The reason is that if every time undo is pressed we need to send all the data of all the pixels to each and every client, it would be too much to transfer. Another

very important reason is that even if the network capacity is extremely large, we cannot do this because we cannot handle the situation that multiple users and drawing simultaneously on the same canvas but each user can only undo his or her own line segments. Therefore if one user need to have the canvas saved in the stack while others are still drawing, it obviously causes a problem that when undo is performed we cannot simply restore the previous canvas.

The second reason is that when a new user enters the application, all the previous line segments need to be sent to him or her in order to initialize the canvas. Then it also causes some difficulties when dealing with data synchronization if other users are still drawing.

Therefore we implemented the multiple layer mechanism, which can be divided into three parts:

- *Base Canvas*: The canvas in the lowermost position, where all the static segments are drawn. Since now we only support single-step undo and redo (multiple-step undo and redo is extremely difficult if we don't have good enough network speed to transfer the entire canvas for each undo and redo), we need to store the segments that are not going to be removed.

- *User's Layers*: Each user has its own layer on which the latest line segment is drawn. When the user press down the mouse button, the corresponding layer will be cleared and the line segment is moved to the base canvas. Undo and redo can easily be implemented in this way: just clearing (undo) and restoring the user's layer will work.

- *Detector Canvas*: Since only the uppermost layer can detect user's mouseevent, we add another layer to the top of all the canvases. We set the mouseevents for this canvas, so that we can detect the events without having to change too much of the webpage.

However, a problem arises with the implementation of multiple canvas. That is the layer overlay. We must have some user's layer above other users' layer and if that user draw a huge, black rectangle on his or her own layer and do not push it to the base canvas by another mouseclick, others will not be able to see what they drew, because their segments will be blocked by the rectangle. To solve this problem, every time a user presses the mouse down, his or her layer would be put above all the other's layers and then this problem is solved. Now whoever draws the last segment will have his or her segment above all the others'.

To efficiently change the arrangement of the layers, jQuery helped us a lot because it provides us a lot of APIs for dynamically changing

the HTML file of the webpage. We can insert a new layer when a new user enters, delete a layer when the corresponding user leaves and change the order of the layers all by one function call.

4.2 Upload and Download

Without upload and download functionalities, the application would not be as practical as it is. The implementation of upload and download actually enables the users to save the work. And if the user has a existing picture to be drawn on, he or she can merely do it within a few mouseclicks and drags.

We can find the buttons for saving and uploading at the top-right corner of the webpage. They are quite user friendly and users will benefit a lot with these functionalities.

The implementation also requires the communication between the client-side and server-side. We need to store the picture in the database and send it to all the other users.

5 Web Frameworks

In this section we will introduce the framework we used for development, including Mojolicious, the Perl framework, HTML5 and jQuery.

- *Mojolicious*: The server side is implemented using the Perl web framework of Mojolicious, which provides some useful APIs for web programming, including WebSocket. Mojolicious works quite efficiently when large amount of data need to be transfered. Since our application is really data-dense, a efficient WebSocket framework is needed and fortunately Mojolicious provides it.

- *HTML5 Canvas*: Thanks to the emerging of HTML5 canvas, which is a nice platform for developing various browser-based graphical applications, our project can start much easier than in the old days when the similar functionalities have to be implemented by Flash of SVG. The canvas mechanism provides a lot of handy APIs that can easily be adopted.

- *HTML5 Websocket*: The new HTML5 also provides built-in support for WebSocket, an efficient and user-friendly communication tool for data transfer. Now the web socket is directly accessible from Javascript. However, one important drawback of the build-in web socket APIs is that they are now available for only a few web browsers, including Google Chrome, Chromium, and Safari, not even Mozilla Firefox (up till now), since the common agree-

ment of HTML5 standard has not been reached yet.

- *jQuery*: On the client side, we made use of the jQuery framework, which reduced the total lines of code and the amount of unnecessary work with it great simplicity, since the library provides us many shortcuts to implement the functionalities which otherwise would take hundreds more lines of javascript code to implement.

- *JSON*: In order to make to data transfer easier to deal with, we also made use of JSON, a simple tool to stringify data and what we need is just to make use of the API built-in in both Javascript and Mojolicious.

6 Knowledge Used from COMP2021

We used several important points that are covered in COMP2021 in doing this project

- *Unix*: We worked on this project entirely on Linux. What we did includes setting up the server, using **vim** to edit the programs, using **git** as the tool for version control and so on.

- *Perl Programming*: The entire sever-side program is written in Perl, through which we obtained a deeper understanding not only about the Perl programming language, but about the script languages as well.

- *HTML and Client-Side Programs*: The most exciting part of this application is done by HTML5 and the client-side programs.

- *CGI Programming*: This application, although not using the CGI model provided by the lecture notes, made use of the CGI-like framework and WebSocket. Huge amount of data transfer between the server side and the client side is done. Without the course materials we would have found it much more difficult to understand the mechanism.

7 Further Development

We think this application is worth further development. We will add some features and improve the appearance in the future and develop it into a fully operational web application.

- *Get a Server*: We will buy or use a free server to install our application. Then we are free to do something big without using

our own machines for temporary testing.

- *Signing Up*: We are going to implement the register function and thus each client can have his or her own workspace to store and manage the works. In this way each user will have an account and some space.

- *Online Saving*: Since each user can have a workspace, we can allocate each of them some storage to store the works.

- *Sharing*: Users will be able to share works with other users.