Question

Alternative design (12 marks)

On the client-side you have been able to choose between React.js, Angular.js, and d3.js , vue.js

I would like you to detail how your application could be built using alternative technologies. For each of the major components of your app:

- server-side framework
- client-side framework

Select an alternative technology and and show (with example code, and a brief explanation of how it works) how a key piece of code from your app could be implemented in the alternative framework.

For example, if your app was developed using React.js, you might select one of the main views from your project, and show how that view could be implemented using Angular.js. For the server framework, you might choose how websockets are implemented in, say, Node.js. (You are not constrained to just frameworks we've talked about in class.)

For the database, you might show how your code would be different if you used a relational database. Or you might show how your code could be implemented using an object-relational mapper (eg, Anorm, JPA, or Slick).

Don't forget to include *both* the code from your project *and* your sample of how that could be implemented in the other framework.

As this assignment is submitted as a PDF, rest assured that your code will be read, not compiled, and I do not expect you to have tested your sample code. Getting the explanation and the fundamentals of the framework you've chosen right is more important than minor syntax errors.

Length: Including code examples, this should be less than 5 pages.

Marking: The server, and client will each be marked on a 1 to 4 scale. Focus on the clarity of your explanation -- your mark here depends on showing that you understand the framework you are presenting and how to design for it.

Extended design (4 marks)

I would like you to consider how your system could be extended, to add a piece of useful functionality.

You should:

- describe what the functionality is and why you think it would be useful to a potential user
- describe what changes would need to be made to your system to enable it

The nature of the change will depend on your feature. For some projects, this might involve an extra API or integrating with another external system. For others, the changes might involve redesigning existing functionality.

Length: 500 words

Client side framework:

We used **AngularJS** for client side and the following code was used for login page:

```
var app = angular.module("myChatApp", []);
app.controller("loginCtrl", function($scope)
{
    $("#startBtn").click(function(e){
        if(!$scope.username || $scope.username=="")
        {
            alert("Please specify a username before you start chat !")
           e.prevantDefault();
        }
        else{
            alert("welcome Mr "+$scope.username);
        }
    })
});
```

The code simply checks if the scope username variable is set when user clicks to join collaborative draw/chat application, if it is then an alert is shown to welcome the user using his/her username, otherwise it asks him/her to specify it.

This would be done using **React.js** components (we will focus on login page only):

```
 class MainComponent extends React.Component
{
    return HTML_CODE;
}

class LoginComponent extends React.Component
{
    constructor(props)
    {
        this.username = null;
    }

    setUsername(e)
    {
        e.preventDefault();
        if(this.ref.username.length > 0)
            this.username = this.refs.login;
    }

    isAuthenticated()
    {
        return this.username != null;
    }

    render()
    (
        if(this.isAuthenticated()){
            return <MainComponent />
         }
         return
            <form onSubmit={ this.setUsername }>
                <input placeholder="Username" ref="username" />
                <button>Start chat</button>
            </form>;
    );
}

React.render(<LoginComponent />, document.getElementById('container'));
```

- MainComponent: to render whiteboard and chat box
- LoginComponent: to render login form if user not authenticated (submitting the form assign the username to the appropriate attribute, the authentication is checked by checking if this attribute is assigned or not), otherwise it should render the MainComponent.

Server side framework:

We used **Express** for server side with **socket.io** to achieve the collaborative drawing and chat and the following code is a snippet from the one used for socket server:

```
	var express = require('express');
	var app = express();
	var server = require('http').createServer(app);
	var socketIO = require('socket.io').listen(server);

	app.use(express.static('public'));

	server.listen(process.env.PORT || 3000);
	console.log('Server started at port 3000');

	app.get('/', function (req, res) {
		res.sendFile(__dirname + '/index.html');
	});

	socketIO.sockets.on('connection', function (socket) {
		connections.push(socket);
		socket.on('disconnect', function (data) {
			if(socket.username){
				// remove user from the list
				users.splice(users.indexOf(socket.username), 1);
				updateConnectionsBox();
			}
			connections.splice(connections.indexOf(socket), 1);
		});

		socket.on('send msg', function (data) {
			socketIO.sockets.emit('new msg', {msg: data, username: 
socket.username});
		});
		socket.on('new usr', function (data, callback) {
			if(!data){
				callback(false);
				return;
			}
			callback(true);
			socket.username = data;
			users.push(socket.username);
			updateConnectionsBox();
```

```
            });
        });
```

Note that each **socket.emit** is sending to each socket instance from the socket client, and **socket.on** is listening for an event (specified as the first parameter) from the socket client.

Sails.js provides a socket client, "*The Sails socket client (sails.io.js) is a tiny browser library that is bundled by default in new Sails apps. It is a lightweight wrapper that sits on top of the Socket.IO client whose purpose is to make sending and receiving messages from your Sails backend as simple as possible.*" - http://sailsjs.com/documentation/reference/web-sockets/socket-client

Sails.js use the concept of group (room), a group is first created base on a model (nodejs module), then on each new client connection we need to subscribe them to the group (using *watch* method from sails socket client), and publish each of their message to the group (using *publishCreate* method)

Following is an equivalent to the use of express socket.io:

- On the socket server (a controller to catch users requests and either subscribe them or publish their messages depending on the method used to request)

```
module.exports = {
    addConv: function(req, res)
    {
        var data_from_client = req.params.all();
        if (req.isSocket && req.method === 'POST') {
            // This is the message from connected client
            // So add new conversation
            Chat.create(data_from_client).exec(function(error, data_from_client) {
                Chat.publishCreate({
                   id: data_from_client.id,
                   message: data_from_client.message,
                   user: data_from_client.user
                });
            });
        } else if (req.isSocket) {
            // subscribe client to model changes
```

```
                Chat.watch(req.socket);
            }
        }
    };
```

- The model being used here is as follows (Chat.js):

```
    module.exports = {
        attributes: {
            user:{
                type:'string'
            },
            message:{
                type:'string'
            }
        }
    };
```

- Finally the user (socket client) would first subscribe to the group using get request then send message via post requests:

```
    io.socket.get('/chat/addconv');

    sendMsg = function() {
        io.socket.post(
            '/chat/addconv/',
            { user: $scope.chatUser, message: $scope.chatMessage }
        );
        $scope.chatMessage = "";
    };
```

Extended design:

Sometimes while chatting publicly in an open room, one may have the need to share something with a specific person, something that simply cannot be published visibly to everyone. An interesting functionality we can provide is to send private messages to a specific user.

To do that we first need to improve the login system with a signup form first to provide a unique id to each user. We push each new socket in a list (e.g socketList) identified by the user id. Then we can provide a route (e.g POST to /send/userid) which will contain the following logic: check if there is a socket for that userid in the socketList, if it’s the case emit a signal to that socket with the private message. Finally, we can make that in the connected users list, one might click on a user name to popup a textarea to send a private message (through the POST request).