10 luglio 2012

Specifica Tecnica

Versione: 1.0

Informazioni sul documento

Nome	Specifica_tecnica.pdf
Versione	1.0
Data di creazione	30 maggio 2012
Data di ultima modifica	10 luglio 2012
Stato	Formale
Uso	Esterno
Distribuzione	Alessandro Zonta, il tutor aziendale e il tutor interno.

Indice

1 Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è quello di fornire una visione della progettazione che si intende attuare nello sviluppo dell'applicazione Aula virtuale.

1.2 Scopo del prodotto

L'applicazione Aula virtuale permette l'autenticazione in un'anticamera dove avviene la scelta di accedere ad una stanza o di crearne una nuova dove poter liberamente parlare, vedersi, chattare e condividere un'area di lavoro. Tale software fa uso del linguaggio HTML5[g] e delle nuove API che mette a disposizione per fornire connettivita tra diversi dispositivi senza utilizzo di plugin[g] o altri software.

1.3 Glossario

Al fine di eliminare ambiguità e incomprensioni, tutti i termini tecnici e le sigle utlizzate nel seguente documento sono riportati nel Glossario, fornito come allegato. Alla prima occorrenza i termini presenti nel glossario verranno rappresentati con la sigla [g] alla fine della parola.

1.4 Riferimenti

1.4.1 Normativi

- Documento *Analisi_dei_requisiti*

1.4.2 Informativi

- Draft WEBRTC
- Web Real Time Communication by Ericcson Labs
- Draft getUserMedia
- jwebsocket.org
- Geolocation API[g] Specification
- PDF.js
- File API

2 Definizione del prodotto

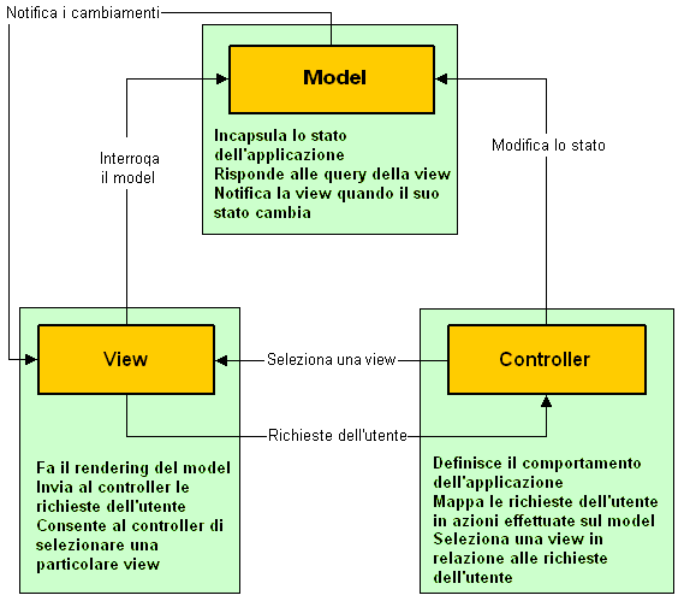

Figura 1: Architettura MVC

Aula virtuale utilizza il pattern architetturale MVC. Tale pattern isola il dominio logico dall'interfaccia utente permettendo lo sviluppo, il testing e la manutenzione indipendente di ciascuno. Esso prevede di strutturare l'applicazione in tre componenti:

- model: incapsula lo stato dell'applicazione, definisce i dati e le regole di business per l'interazione con questi ultimi. Espone alla View ed al Controller rispettivamente le funzionalità per l'accesso e l'aggiornamento;
- view: gestisce la costruzione dell'interfaccia grafica (GUI);
- controller: trasforma le interazioni dell'utente della View in azioni eseguite sul Model. Mantiene la consistenza tra i dati mostrati dalla View e quelli presenti nel Model.

La relazione tra view e controller può essere di due tipi: push o pull. Secondo il modello push le azioni dell'utente vengono interpretate dal model il quale genere l'informazione da inserire nella view. Invece per il modello pull la view accederà al controller per prendersi le informazioni di cui ha bisogno per visualizzare l'output richiesto. La maggior parte dei web-framework implementa uno solo dei due metodi. Nel nostro caso faremo riferimento al push model. Lo scopo di questo pattern è disaccopiare il più possibile le macro-componenti dell'applicazione. L'uso di tale pattern porta a ridurre la complessità nel design architetturale e ad aumentare la flessibilità e la manutenibilità del codice.

3 Tecnologie

Questa sezione descriverà i linguaggi e le nuove API utilizzate nello sviluppo dell'applicazione.

3.1 Linguaggi

3.1.1 HTML5

L'HTML5 è un linguaggio di markup per la progettazione delle pagine web attualmente in fase di definizione (bozza) presso il World Wide Web Consortium. Essendo il software un'applicazione online esso è interamente basato su questa tecnologia, che ne permette l'esecuzione sulla maggior parte dei browser attualmente nel mercato.
HTML5 introduce e consente di utilizzare diverse funzionalità tra cui i WebSocket, le MediaAPI, la geolocalizzazione e l'offline storage.

3.1.2 Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web. L'applicazione è interamente fondata su questa tecnologia perchè permette di interagire con le API messe a disposizione da HTML5 direttamente sul browser del client senza nessuna imterazione da parte di un server.

3.1.3 JSON

JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi. JSON è basato su due strutture:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record[g], uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo;
- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

Queste sono strutture di dati universali. Virtualmente tutti i linguaggi di programmazione moderni li supportano in entrambe le forme. E' sensato che un formato di dati che è interscambiabile con linguaggi di programmazione debba essere basato su queste strutture.

3.2 API

3.2.1 WebSocket

WebSocket sono un set di API nato per permettere ai browser e server di parlarsi in maniera asincrona e senza bisogno dell'interazione dell'utente. La comunicazione asincrona è quella comunicazione in cui il mittente invia il messaggio e poi continua la propria esecuzione; asincrona sta proprio a significare l'asincronicità che vi è tra l'invio di un messaggio e la risposta al messaggio stesso.
Per l'applicazione è stato scelto di utilizzare jWebSocket, un'implementazione open source in Java e Javascript del protocollo WebSocket con alcune estensioni molto interessanti. Il package jWebSocket contiene:

- *jWebSocket Server*: un WebSocket server scritto interamente in Java per streaming[g] di dati server-to-client(S2C) e controllo comunicazioni client-to-client (C2C);
- *jWebSocket Clients*: un WebSocket client scritto interamente in JavaScript con la possibilità di utilizzare diversi sottoprotocolli, sessioni e timeout-management. Non richiede l'installazione di plug-in nel broser.

Il server fornito da jWebSocket fornisce già di default le caratteristiche richieste dall'applicazione, come fornire l'accesso al client, effettuare il broadcast[g] dei messaggi per una chat e fornire la divisione in stanze separate delle comunicazioni. Una caratteristica molto importante che ha portato all'adozione di jWebSocket è che è possibile integrarlo in maniera molto semplice ed efficace in applicazioni web esistenti, come ambienti Tomcat. L'unica richiesta è che sia installata nel sistema Java. Molto interessante è anche la possibilità che jWebSocket server fornisce di poter scrivere dei plug-in, dei listener o dei filtri e integrarli con molta facilità nel server. Nell'applicazione questa possibilità non è stata sfruttata dato che le funzionalità richieste erano già presenti in un'installazione di default del server.

Per il client la situazione è un attimo diversa. Con jWebSocket vengono fornite le librerie più importanti, le quali contengono i metodi basilari per la connessione e autenticazione, ma è necessario scrivere a mano tutti i metodi che permettono la comunicazione tra il client e il server. Nell'applicazioni sono state usate due librerie:

- *jWebSocket.js*: questa è la libreria più importante dato che contiene tutti i metodi necessari per stabilire una connessione con il server. Qualsiasi applicazione che voglia usare jWebSocket deve includere questa libreria;
- *jwsChannelPlugIn.js*: questa libreria fornisce i metodi per poter avere una divisione in stanze per le comunicazioni. Viene fornita con il jWebSocket Clients dato che nel server è già implementata la possibilità di suddivisione in stanze, e permette una facile implementazione da parte del programmatore nelle proprie applicazioni.

jWebSocket.js

Istanziazione di un jWebSocket Client:

```
if( jws.browserSupportsWebSockets() ) {
 jWebSocketClient = new jws.jWebSocketJSONClient();
 // Optionally enable GUI controls here
} else {
```

```
 // Optionally disable GUI controls here
 var lMsg = jws.MSG_WS_NOT_SUPPORTED;
 alert( lMsg );
}
```

Creare un'istanza di jWebSocketClient è molto semplice. jWebSocket fornisce la classe jWebSocketJSONClient la quale contiene i metodi per connettersi, disconnettersi e scambiare messaggi con il server usando il formato JSON. Questa classe è inclusa in un specifico namespace per non andare in conflitto con altri framework.

Autenticazione sul jWebSocket server:

```
log("Connecting to"+lURL+"and logging in as'"+gUsername+"'...");
var lRes = jWebSocketClient.logon( lURL, gUsername, lPassword, {
 // OnOpen callback
 OnOpen: function( aEvent ) {
  log( "jWebSocket connection established." );
 },
 // OnMessage callback
 OnMessage: function( aEvent, aToken ) {
  log("jWebSocket'"+aToken.type+"'token received,
   full message:'"+aEvent.data+"'");
 },
 // OnClose callback
 OnClose: function( aEvent ) {
  log( "jWebSocket connection closed." );
 }
});
```

Per iniziare la connessione tra il client e il server viene usato il metodo *logon* presente nella libreria jWebSocket.js. Questo metodo connette il client al server passandoli username e password per l'autenticazione. Il server assegnerà un ID unico per un client, così se un utente si autentica al server con diversi browser dalla stessa macchina apparirà come più persone.

Spedire un messaggio con jWebSocket:

```
// lMsg is a string
if( lMsg.length > 0 ) {
 var lRes = jWebSocketClient.broadcastText(
  "",   // broadcast to all clients (not limited to a certain pool)
  lMsg  // broadcast this message
 );
 if( lRes.code != 0 ) {
  // display error
 }
}
```

Se la connessione è avvenuta con successo un client può spedire un messaggio ad un altro preciso client utilizzando il metodo *send* oppure spedirlo a tutti gli altri presenti usando il metodo *broadcast*; entrambi appartenenti alla libreria jWebSocket.js. L'azione di spedizione di un messaggio è sempre non bloccante, i metodi *send* e *broadcast* non attendono che la spedizione ritorni un avviso di successo, che è opzionale.

Processare messaggi in arrivo:

```
// OnMessage callback
OnMessage: function( aEvent, aToken ) {
 log( "<font style='color:#888'>jWebSocket '" +
 aToken.type + "' token received, full message: '" +
 aEvent.data + "'</font>"
 );
}
```

I messaggi dal server al client sono spediti asincronamente. Pertanto la classe jWebSocketClient fornisce l'evento *OnMessage*. Basta semplicemente aggiungere un listener per questo evento nell'applicazione il quale all'arrivo di un messaggio lo processerà come desiderato.

Chiudere la connessione:

```
if( jWebSocketClient ) {
 jWebSocketClient.close();
}
```

Sia il server che il client possono terminare la connessione su richiesta. Per il client esiste il metodo *close* fornito dalla classe jWebSocketClient. Il server termina automaticamente la connessione dopo un certo periodo di inattività nella linea. Se questo succede il client riceve un evento intercettabile dal listener *OnClose* il quale disconnetterà il client permettendo all'applicazione di comportarsi nel modo giusto. In ogni caso il timeout del server può essere configurato a piacimento oppure si può avvisare il server che deve rimanere connesso con il client anche se non c'è attività tra di loro oppure si può fornire il client di un metodo che lo ricconnette al server se questo si disconnette.

jwsChannelPlugIn.js

Creare un canale:

```
this.log("Creating channel '" + VchannelId + "'...");
var ris = this.lWSC.channelCreate(
 VchannelId,
 VchannelName,
 {       isPrivate: VisPrivateValue,
         isSystem: VisSystemValue,
         accessKey: VaccessKeyValue,
         secretKey: VsecretKeyValue
 }
 );
this.log(this.lWSC.resultToString(ris));
```

Utilizzando la libreria jwsChannelPlugIn.js è possibile creare delle stanze separate dove poter accedere e effettuare operazioni senza che le persone presenti nelle altre stanze se ne accorgano. Grazie a questa libreria è molto semplice, basta inserire un nome, un id e una password che serve per tenerne traccia, accederci o eliminarla e una password per permettere agli utenti di pubblicare in quel canale.

Accedere ad un canale:

```
this.subscribeChannel = function(channel,accessKey) {
 this.actualChannel=channel;
 this.actualAccessKey=accessKey;
 this.log("Subscribing at channel '" + this.actualChannel + "'...");
 var ris = this.lWSC.channelSubscribe(this.actualChannel,this.actualAccessKey);
 this.log(this.lWSC.resultToString(ris));
};
```

Molto semplice è anche l'accesso al canale. Formalmente si parla di canali, perchè il server divide le comunicazioni in canali. É il programmatore che può usare questa distinzione in canali per creare stanze separate tra loro per fare comunicare i client in privato.

Spedire un messaggio in un canale:

```
this.publish = function(message,selettore,controllore){
 this.log("Publishing to channel '" + this.actualChannel + "'..." );
 var ris = this.lWSC.channelPublish(this.actualChannel,message,{
  field1: selettore,
  field2: controllore
 });
 this.log(this.lWSC.resultToString(ris));
};
```

Naturalmente è possibile inviare messaggi solo dal canale con cui si ha l'accesso.

3.2.2 getUserMedia

Queste API permettono al browser di catturare l'audio e il video della webcam integrata (o installata) e visualizzarlo nella pagina grazie ad un nuovo tag <*video*> introdotto con l'HTML5. Questa API è recentissima pertanto solo in Chrome e Opera funziona attualmente, e solo nelle versioni beta. In Chrome è necessario abilitare un flag specifico altrimenti non funziona.
Per ottenere l'accesso alla webcam o al microfono è necessario richiedere all'utente di confermare il loro utilizzo da parte della pagina web. Nella richiesta è possibile specificare a quale media vogliamo accede: o solo il video o il video e l'audio. Ottenuto l'accesso otteniamo un oggetto *LocalMediaStream* che rappresenta la webcam dal quale possiamo ottenere un *Blob URL* che, se assegnato ad un tag video, ci permette di visualizzare lo stream della webcam. Un Blob URL rappresenta sostanzialmente un insieme di dati grezzi immutabili.

```
window.navigator.webkitGetUserMedia({audio:true,video:true},function(strm){
        var player = document.createElement("video");
        player.setAttribute("id", "selfvideo");
        player.width = 160;
        player.heigth = 120;
        player.src = webkitURL.createObjectURL(strm);
        player.autoplay = true;
        document.getElementById("video").appendChild(player);
},function (){
        alert("Webcam non disponibile");
});
```

3.2.3 Geolocation API

La funzionalità offerta da Geolocation API è abbastanza semplice: definire una struttura dati atta a contenere vari dati geospaziali e impostare alcune funzioni di accesso a tali dati. Nessuna specifica viene definita in merito ai meccanismi che il browser deve utilizzare per recuperare questi dati, ogni piattaforma è infatti tenuta ad utilizzare al meglio le informazioni provenienti dal device. Su di un dispositivo mobile di ultima generazione avremo quindi un set di coordinate provenienti dal sensore GPS, mentre su di un portatile potremo avvalerci del posizionamento legato all'ip della connessione internet.
Esistono solo due metodi, molto simili tra loro ma con una semplice differenza: *getCurrentPosition* restituisce la posizione di dove attualmente ci si trova, *watchPosition* restituisce la posizione ogni volta che quest'ultima cambia.

```
navigator.geolocation.getCurrentPosition(inCasoDiSuccesso
    ,opzInCasoDiErrore,opzioni);
navigator.geolocation.watchPosition(inCasoDiSuccesso
    ,opzInCasoDiErrore,opzioni);
```

La sintassi dei metodi è semplice, il primo argomento contiene il riferimento ad una funzione da eseguire in caso di successo, il secondo argomento è opzionale e punta ad una funzione da esegure in caso di errore. L'ultimo parametro anch'esso facoltativo può essere usato per specificare alcune opzioni utilizzando la struttura opzione1:valore1,... Sono disponibili tre opzioni:

- *enableHighAccuracy (true/false)*: questo flag può essere utilizzato per notificare allo user-agent la necessità o meno di ottenere dati il più accurati possibile;
- *timeout (millisecondi)*: l'opzione rappresenta il tempo massimo concesso al browser per recuperare la posizione dell'utente;
- *maximuAge (millisecondi)*: indica al browser di effettuare una ricerca preventiva nella cache di un dato geospaziale non più vecchio dei millisecondi specificati. Se disponibile tale dato verrà restituito come posizione corrente, altrimenti verrà eseguita la procedura classica.

La funzione invocata in caso di successo deve accettare un singolo argomento, un oggetto di tipo Position che contiene tutte le informazioni recuperate così come un timestamp delle data e ora di recupero.

```
inCasoDiSuccesso = function(position){
 alert("Posizione delle:"+position.timestamp.getHours()+":"+
 position.timestamp.getMinutes() + "n" +
 "Accuratezza delle coordinate:"+position.coords.accuracy+"mt;n"+
 "Latitudine: " + position.coords.latitude + " gradi; n" +
 "Longitudine: " + position.coords.longitude + "gradi; n" +
 "Accuratezza dell'altezza:"+position.coords.altitudeAccuracy+"mt;n"+
 "Altezza: " + position.coords.altitude + " mt; n" +
 "Direzione: " + position.coords.heading + " gradin " +
 "(0 = Nord, 90 = Ovest, 180 = Sud, 270 = Est);n" +
 "Velocita: " + position.coords.speed + " m/s;"
 );
}
```

Queste API sono state utilizzate in una prima versione dell'applicazione per farsi restituire l'orario e la posizione di quando veniva effettuata una fotografia.

3.2.4 PeerConnection API

Queste API sono molto potenti e permettono di scambiare dati tra due browser direttamente senza uso di un server come con i WebSocket. I pregi di una comunicazione di questo tipo è maggior efficienza nell'utilizzazione della rete e una latenza minore. In ogni caso è necessario uno scambio di informazioni iniziale tramite un server per poi permettere ai due browser di connettersi direttamente tra di loro e trasferirsi informazioni. Nell'applicazione come canale di comunicazione è stato utilizzato il server fornito da jWebSocket.

Essendo una comunicazione p2p[g](Peer-to-peer) quella instaurata tra i due browser è necessario fornire al momento della creazione della comunicazione l'indirizzo di un server STUN(Session Traversal Utilities for Network Address Translators)/TURN(Traversal Using Relays around NAT). Se i peer sono nella stessa rete non servirebbe neanche fornire l'indirizzo di un server STUN/TURN però in questo modo si uscirebbe dagli standard definiti. Il protocollo STUN permette alle applicazioni dietro un NAT[g](network address translation) di scoprire la presenza del traduttore di indirizzi(NAT) e di ottenere l'IP pubblico mappato e il numero della porta che il NAT ha allocato per la connessione UDP con l'indirizzo remoto. Il protocollo richiede la presenza di un server (STUN server) situato dalla parte opposta (parte pubblica) del NAT. Dopo che il client ha scoperto il suo indirizzo esterno, può usarlo per connettersi e comunicare con un altro peer direttamente comunicandoli il suo indirizzo esterno al NAT piuttosto che quello interno che per definizione è privato e quindi non raggiungibile da altri peer tramite internet.

Una volta che ognuno dei due peer viene a conoscenza del proprio indirizzo di rete esterno devono stabilire una comunicazione tra di loro. Per fare questo viene utilizzata la tecnica dell' *UDP hole punching*. UDP Hole punching consente di stabilire delle connessioni tra due host attraverso uno o più NAT. Per funzionare, questa tecnica richiede che i due host A e B iniziano una trasmissione UDP con un server S ed essendo entrambi appartenenti a reti private interviene il NAT che assegna alla trasmissione UDP delle porte esterne temporanee. Il server S controlla se le porte sorgenti usate da A e B sono le stesse usate per spedire il messaggio (se il NAT crea le porte randomicamente l'UDP hole punching non può venire effettuato) e se lo sono A e B scelgono due nuove porte X e Y per instaurare la comunicazione e avvisano S di questa scelta. S allora avvisa A di inviare un pacchetto UDP verso la porta Y di B e B di connettersi con la porta X di A, in questo modo ora i due host A e B possono comunicare direttamente tra di loro senza problemi.

Il metodo che permette la creazione della comunicazione è semplice, accetta solamente due parametri. Il primo è l'indirizzo del server STUN/TURN e il secondo è la funzione che viene richiamata per poter procedere con il primo scambio di informazioni tramite jWebSocket. Per lo scambio die dati si utilizza il protocollo SDP che serve per arrivare ad un comune accordo su dettagli riguardanti l'imminente trasmissione p2p dei video. In pratica tramite questo protocollo vengono stabiliti vari parametri quali codec e l'indirizzo della trasmissione video. Poi bisogna dichiarare degli ascoltatori che richiamano funzioni quando si verifica un determinato evento. Per esempio quando la connessione è attiva tra i due peer e uno dei due invia lo stream catturato dalla webcam, l'altro peer riceve un evento onAddStream il quale permette al peer di creare al volo un tag video dove mostrare lo stream in arrivo.

```
PC=new window.webkitDeprecatedPeerConnection(this.serverConfig,function(sdp){
  console.log("to send SDP stuff to id: "+ID);
  jWebSocket.sendText(ID,{"scopo":"realTcomm","motivazione":"SDP","dato":sdp});
```

```
});
PC.addStream(strm);
PC.onaddstream = onAddStream;
PC.onconnecting = onConnecting;
PC.onopen = onOpen;
PC.onclose = onClose;
PC.onerror = onError;
PC.onremovestream = onRemoveStream;
function onAddStream(evt){
 console.log("incoming stream");
 var player = document.createElement("video");
 player.width = 160;
 player.heigth = 120;
 player.src = webkitURL.createObjectURL(evt.stream);
 player.autoplay = true;
 document.getElementById("video").appendChild(player);
};
function onConnecting(evt){
 chan.log("PC connecting");
};
function onOpen(evt){
 chan.log("PC opened");
};
function onClose(evt){
 chan.log("PC closed");
};
function onError(evt){
 chan.log("PC error");
};
function onRemoveStream(evt){
 chan.log("stream removed");
};
```

É stato provato che è possibile creare multiple PeerConnection da un host a molti altri, solo che essendo ancora molto acerbe come api, non contemplano ancora dei metodi per otimizzare alcuni aspetti del flusso video causando sforzi notevoli alla cpu di pc non di ultima generazione.

Come si può notare dal codice sopra, sono state usate le *webkitDeprecatedPeerConnection* dato che attualmente sono le uniche implementate da Chrome. La bozza ufficale del w3c[g] è aggiornata al 30 maggio 2012 ma le nuove specifiche non sono ancora state implementate. Per l'applicazione si seguono le metodologie specificate nella bozza aggiornata al 27 aprile 2012. Questo perchè queste api sono molto giovani, la prima bozza del w3c è stata pubblicata nel gennaio 2012 e attualmente queste api funzionano solamente su Chrome Canary anche perchè l'autore iniziale di queste bozze è un dipendente della Google.

3.2.5 Web SQL[g] database API

Tra le varie prove per l'applicazione è stato testato pure il salvataggio di immagini nella memoria del browser utilizzando le Web SQL database API. Queste API portano l'SQL anche nella parte client e contengono tre importanti metodi per amministrare il tutto:

- *openDatabase*
- *transaction*
- *executeSql*

Attualmente la compatibilità di queste API è con Safari, Chrome e Opera. Firefox non implementa queste API perchè Mozilla ritiene che esistano migliori implementazioni per il salvataggio client-side.

Creare un nuovo database è semplice. Se si prova ad aprire un database inesistente le API lo creeranno da zero e inoltre non bisogna preoccuparsi di chiuderlo al termine delle operazioni. Per creare e aprire un database si utilizza la seguente riga di codice

```
var db = openDatabase('mydb', '1.0', 'my first database', 2 * 1024 * 1024);
```

Vengono passati 4 parametri al costruttore:

- *nome*
- *versione*
- *descrizione*
- *dimensione stimata del database*

Dopo aver creato o aperto un database si può iniziare ad operare con i dati. Per fare questo dobbiamo creare una transazione. Questo perchè le transazioni contengono le istruzioni SQL e forniscono all'applicazione la possibilità di fare *rollback*[g]. Questo significa che se una transazione fallisce, i cambiamenti non vengono effettuati nel database come se la transazione non fosse mai partita.

```
db.transaction(function (tx) {
 // here be the transaction
 // do SQL magic here using the tx object
});
```

Per eseguire le query SQL si utilizza il comando *executeSql* sia per leggere che per scrivere. Se una query fallisce le sucessive presenti nella stessa transizione non vengono eseguite.

```
db.transaction(function (tx) {
 tx.executeSql('CREATE TABLE foo (id unique, text)');
});
```

3.2.6 Canvas 2D API

Con l'avvento dell'HTML5 è stato implementato un tag chiamato canvas che permette di disegnare sul browser. Il canvas per definizione è un elemento invisibile grande 300x150px (di default) compatibile con tutti i browser. La prima cosa da fare è ottenere tramite javascript il *drawing context*, il quale ci permette di avere un accesso diretto al canvas e quindi disegnarci sopra.

```
<script>
.// difinito un canvas con id="c"
 // declare the two variables at once using a comma
 var canvas = document.getElementById("c"),
 context = canvas.getContext("2d");

 // now you're ready to draw
</script>
```

Le API forniscono numerosi metodi per disegnare e settare proprietà utili al disegno come *fillStyle* e *strokeStyle* per settare il colore con cui si andrà a disegnare forme e linee. I colori sono gli stessi che si usano per il CSS[g] come rgb() o rgba(). Vengono forniti anche metodi per disegare forme ma nell'applicazione attualmente non vengono usati.
Per disegnare a mano libera si utilizzano le canvas paths. Per iniziare a disegnare bisogna chiamare *beginPath()* e poi disegnare liberamente sul canvas. Una volta disegnato bisogna chiamare *closePath()* altrimenti nulla di quanto disegnato sarà visibile.

```
// Set the style properties.
context.fillStyle   = '#00f';
context.strokeStyle = '#f00';
context.lineWidth   = 4;

context.beginPath();
// Start from the top-left point.
context.moveTo(10, 10); // give the (x,y) coordinates
context.lineTo(100, 10);
context.lineTo(10, 100);
context.lineTo(10, 10);

// Done! Now fill the shape, and draw the stroke.
context.fill();
context.stroke();
context.closePath();
```

3.2.7 File API

Finalmente con HTML5 viene fornito un modo standard per interagire con i file locali tramite la specifica API File. Queste API permettono ad una Web Application di richiedere uno spazio dedicato per salvare i dati in maniera temporanea o persistente, oppure di leggere dei dati presenti nel filesystem. La specifica prevede diverse interfacce per l'accesso ai dati locali:

- *File* - un singolo file; vengono forite informazioni come nome, grandezza, tipologia e riferimento al file.
- *FileList* - un arrey di oggetti File.
- *Blob* - Permette la divisione del file in intervalli uguali.

L'interfaccia FileReader, utilizzata in combinazione con le strutture di dati descritte sopra, permette la lettura di file in modo asincrono tramite la gestione degli eventi Javascript.
La forma più comune per caricare un file è usare la metodologia standard *¡input type='file'¿*.

Javascript ritorna la lista dei dei file selezionati all'interno dell'oggetto FileList. Un esempio emplice che fa vedere le modalità di caricamento di un file. Con l'attributo multiple sull'input tag si permette la selezione multipla dei file. Per ogni file caricato poi si stampa il nome, la tipologia e altre informazioni semplici.

```
<input type="file" id="files" name="files[]" multiple />
<output id="list"></output>

<script>
 function handleFileSelect(evt) {
  var files = evt.target.files; // FileList object

  // files is a FileList of File objects. List some properties.
  var output = [];
  for (var i = 0, f; f = files[i]; i++) {
    output.push('<li><strong>',escape(f.name),'</strong>(',f.type||'n/a',')-',
      f.size, ' bytes, last modified: ',
      f.lastModifiedDate?f.lastModifiedDate.toLocaleDateString():'n/a',
      '</li>');
  }
  document.getElementById('list').innerHTM ='<ul>'+output.join('')+'</ul>';
}

  document.getElementById('files').addEventListener('change',
    handleFileSelect,false);
</script>
```

Dopo aver ottenuto il riferimento al file si istanzia l'oggetto FileReader per leggere il contenuto in memoria. Al termine del caricamento avviene l'evento onload che, intercettato dal listener corretto, permette di leggere l'attributo result dell'oggetto FileReader che può essere utilizzato per accedere ai dati del file.
FileReader include quattro opzioni per leggere asincronamente un file:

- *FileReader.readAsBinaryString(Blob/File)* - La propietà result conterrà i dati del file come una stringa binaria. Ogni byte è rappresentato come un intero nel range [0..255].
- *FileReader.readAsText(Blob/File, opt_encoding)* - La proprietà result conterrà i dati del file come una stringa di testo. Di default la stringa è codificata in UTF-8. Per differenti codifiche si può usare il parametro opzionale per specificare un differente formato.
- *FileReader.readAsDataURL(Blob/File)* - La proprietà result conterrà i dati del file codificati come un Data URL.
- *FileReader.readAsArrayBuffer(Blob/File)* - La proprietà result conterrà i dati del file all'interno di un oggetto ArrayBuffer.

Una volta che uno di questi metodi di lettura è chiamato sull'oggetto FileReader è possibile monitorare i progressi utilizzando i listener onLoadStart, OnProgress, OnLoad,OnAbort, OnError e OnLoadEnded.
Nel progetto le File API sono utilizzate per leggere un file PDF e caricarlo in memoria utilizzando il metodo *FileReader.readAsArrayBuffer(Blob/File)*.

3.3 Librerie

Nel progetto è stata utilizzata una libreria open source ancora in via di sviluppo ma molto innovativa.

3.3.1 PDF.js

Questa libreria permette la visualizzazione di file PDF nel browser senza necessità di plug in aggiuntivi perchè è scritta interamente in javascript. Sta venendo utilizzata anche da Mozilla per il PDF reader integrato che nelle ultime versioni di Firefox è presente. Per utilizzarla è necessario passarle un file ArrayBuffer contenente i dati del PDF da visualizzare. Per questo è stato scelto di leggere i dati con il metodo *FileReader.readAsArrayBuffer(Blob/File)* fornito dalle File API. La libreria si occupa della traformazione e visualizza ogni singola pagina all'interno di un canvas definito dall'utente.
Il funzionamento è molto semplice. Dopo aver incluso la libreria nell'applicazione basta passare l'ArrayBuffer o l'url del file PDF all'oggetto PDFJS che si occupa del lavoro principale e salva il file pdf all'interno di un oggetto. Questo oggetto contiene il numero di pagine totali e grazie ai metodi forniti è possibile renderizzare una singola pagina sapendo solo il numero della stessa.

```
pageNum=1;
PDFJS.getDocument(arrayBuffer/url).then(function getPdfHelloWorld(_pdfDoc) {
  pdfDoc = _pdfDoc;
  renderPage(pageNum); //visualizzo prima pagine pdf
});

this.renderPage = function(num) {
 // Using promise to fetch the page
 pdfDoc.getPage(num).then(function(page) {
   var viewport = page.getViewport(scale);
   viewport.height=200;
   viewport.width=200;
 // Render PDF page into canvas context
 var renderContext = {
   canvasContext: ctx,
   viewport: viewport
 };
 page.render(renderContext);
 });
};
```

Questa libreria purtroppo è ancora in via di sviluppo ed è stato notato che a volte la pagina pdf che viene renderizzata nel canvas risulta essere storta. Non succede sempre, si presenta in modo randomico però è stato notato che se la transizione da una pagina all'altra avviene con calma e non in fretta tutto viene visualizzato correttamente.

4 Aula virtuale

Come definito nei requisiti l'applicazione Aula Virtuale si divide in tre visualizzazioni principali per consentire all'utente una maggiore chiarezza e separazione dei contenuti.

- *Pagina principale*
- *Sala d'attesa*
- *Aula*

4.1 Pagina Principale

Figura 2: Pagina principale

La pagina principale visibile in figura 2 presenta una semplice interfaccia dove viene richiesto un nome utente per poter accedere alle funzionalità dell'aula.

4.2 Sala d'attesa

Figura 3: Salana d'attesa

Una volta effettuato l'accesso si presenta la schermata in figura 3 che permette di scegliere a quale stanza accedere oppure la possibilità di aggiungere o eliminare una stanza. Al momento della creazione di una nuova stanza è possibile anche settare una password per limitare l'accesso a quella particolare stanza. Al click sopra una stanza per accederci viene richiesta la qualifica, cioè se si ha l'intenzione di accedere come professore o come studente.

4.3 Aula professore

Figura 4: Aula Visione Professore

Se si ha scelto di accedere come professore e il sistema ha autorizzato tale accesso (è concesso un solo professore alla volta) si ha accesso all'aula vera e propria. Come mostrato in figura 4 a sinistra è presente un grande canvas che funge da lavagna condivisa. All'inizio solo il professore ha possibilità di disegno, uno studente per poter scrivere deve aver confermato la propria presenza e chiesto permesso al professore. Sotto il canvas sono disponibili una serie di bottoni per interagire con esso. Tutto a destra in caso di schermi larghi oppure sotto al canvas si trovano 3 widget con le funzioni più importanti. Il primo visualizza lo stato della connessione e mette a disposizione due bottoni per il logout e l'uscita dalla stanza. Il secondo widget mostra gli utenti presenti nella stanza in quel momento ed il terzo widget contiene la chat, gli eventuali video quando il professore trasmette e quando un utente alza la mano per parlare e i tre bottoni che permettono l'acquisizione del video e il conseguente streaming p2p, lo stop di tale azione e il caricamento del file pdf nel canvas. Da notare che questi tre widget sono draggabili ovunque nella pagina web (funziona solo per la versione pc). Al di sotto di tutto compaiono le fotografie inviate ogni tot secondi da parte degli studenti che hanno confermato la loro presenza.

4.4 Aula studente

Se si è acceduti come studente il canvas condiviso rimane alla sinistra ma non si possiedono ancora i diritti per scriverci, come visibile in figura 5. Anche lo studente vede tre widget draggabili. I primi due sono uguali a quelli che sono a disposizione per il professore, il terzo ha diverso i bottoni perchè lo studente può solamente confermare la presenza o abbandonare il posto. Una volta che la conferma è stata inviata è possibile per lo studente alzare la mano e se il professor è libero e accetta avviene l'invio del video a tutti quanti e lo studente ottiene il permesso di scrivere sulla lavagna condivisa.

Figura 5: Aula Visione Studente

5 Realizzazione Aula virtuale

In questo capitolo viene spiegato come si è giunti alla realizzazione dell'applicazione Aula virtuale. Questa applicazione non è nata subito pensando a ciò che attualmente fa. Infatti sono state esplorate certe potenzialità e novità dell'HTML5 con altre due mini applicazioni precedenti.

5.1 PhotoBook

Come primo passo è stata creata un'applicazione semplice che subito al caricamento cercava di acquisire il video della webcam. Una volta acquisito questo veniva mostrato nella pagina e, alla pressione di un tasto si poteva scattare una foto (non una reale foto con la fotocamera ma uno screenshot del stream catturato dalla webcam). Questa foto, in formato PNG, veniva salvata in un database dal browser per permette all'utente, nei futuri accessi, di ritrovare le foto scattate. Ogni volta che si scattava una foto veniva anche rilevata la posizione gps di dove si trovava il dispositivo. Era stata scelta la rilevazione con massima precisione, con un timeout di 60 secondi e un'età massima di 30 secondi. Questo per risparmiare un pò di batteria nei dispositivi mobile pur mantenendo una buona qualità di rilevazione. Nell'applicazione sono presenti due modalita di scatto e salvataggio della foto; La prima è differenziando le due azioni e quando si scatta la foto si deve premere il tasto save per salvare la stessa nel database. L'altra modalita accorpa le due funzioni facendo si che ogni scatto venga salvato.

5.2 Peer

Per testare le potenzialità dello streaming p2p della webcam e verificare quali metodi sono stati implementati dato che le bozze del w3c sono ancora in grandi lavori, è stata creata una semplice applicazione che consentiva lo streaming p2p tra più client.
Come spiegato nella sezione 12 per lo scambio delle informazioni riguardanti lo streaming

è necessario in ogni caso un server o un servizio adibito allo scambio di messaggi. Per questa implementazione è stato usato un protocollo sviluppato dai Ericsson Labs chiamato warp che tramite un server tenuto da loro permette uno semplice scambio di informazioni. Il funzionamento è similare a quello dei jWebSocket. Caricata la pagina, si stabiliva un contatto con il server warp che restituiva un id numerico ad ogni pagina; bastava sapere l'id di un altro client, inserirlo nell'apposito box e cliccando un bottone avveniva la connessione con lo scambio di informazioni per poter iniziare una sessione di streaming dalle proprie webcam. Con questa applicazione è stato testata pure la possibilità di avere connessioni multiple da uno stesso client a altri perchè la documentazione online a riguardo era scarna e a volte alcune fonti erano in contraddizione tra loro.

5.3 Aula Virtuale

Dopo queste varie prove si è giunti al punto di sviluppare la vera applicazione Aula Virtuale. Si è iniziato con il creare una connessione tra server e client grazie ai jWebSocket. Una volta compreso il sistema si è organizzato la divisione in stanze utilizzando la libreria apposita da parte client. Una volta che l'accesso a stanze divise è stato implementato si è passato alla creazione della chat sempre tramite i websoket. A questo punto si è implementato l'inseriemnto dell'username per ogni client per permettere la distinzione tra loro dato che jWebSocket assegna un id diverso per client e per l'utente è meglio riconoscersi per nome che per numero identificativo.
Prendendo spunto l'applicazione Peer si è implementato lo streaming p2p della webcam tra tutti i client connessi alla stessa stanza utilizzando però come protocollo di scambio dati jWebSocket dato che la comunicazione con il server è già attiva. Questo si è rilevato molto semplice dato che è bastato aggiungere al metodo di ricezione qualche controllo per capire che tipo di richiesta era stata effettuata, se riguardava la chat o lo scambio d'informazione per la comunicazione p2p.
La libreria che permette al canvas di eseguire le azioni opportune è stata inizialmente sviluppata dal secondo stagista presente in azienda. Questa libreria però è dovuta essere abbastanza riadattata al contesto d'uso di questa applicazione aggiungendo e modificando metodi per permettere la condivisione delle azioni tra tutti gli utenti connessi. Inoltre sono state aggiunte nuove funzionalità per rendere più appagante l'esperienza d'uso del canvas.
A questo punto si è deciso di permettere la scrittura solo al professore e in caso allo studente quando fa richiesta alzando la mano; quindi si è implementato la divisione di accesso tra professore e studente con la limitazione di un unico professore per sessione ma la possibilità per il professore di assegnare il suo ruolo ad uno studente connesso alla stanza perdendo di fatto il potere del professore.
Si è anche sviluppata la parte relativa alla segnalazione di presenza. Ogni studente deve confermare la presenza inviando automaticamente, previa richiesta, una foto catturata dalla webcam al professore e agli altri studenti. In questo modo il professor epuò verificare l'effettiva presenza degli studenti e gli studenti possono vedersi mentre seguono la lezione.
Il professore inoltre ha la possibilità di caricare un PDF preso dal file system e di visualizzare le pagine nel canvas per poterle spiegare, vedere con gli studenti.

6 Specifica delle Componenti

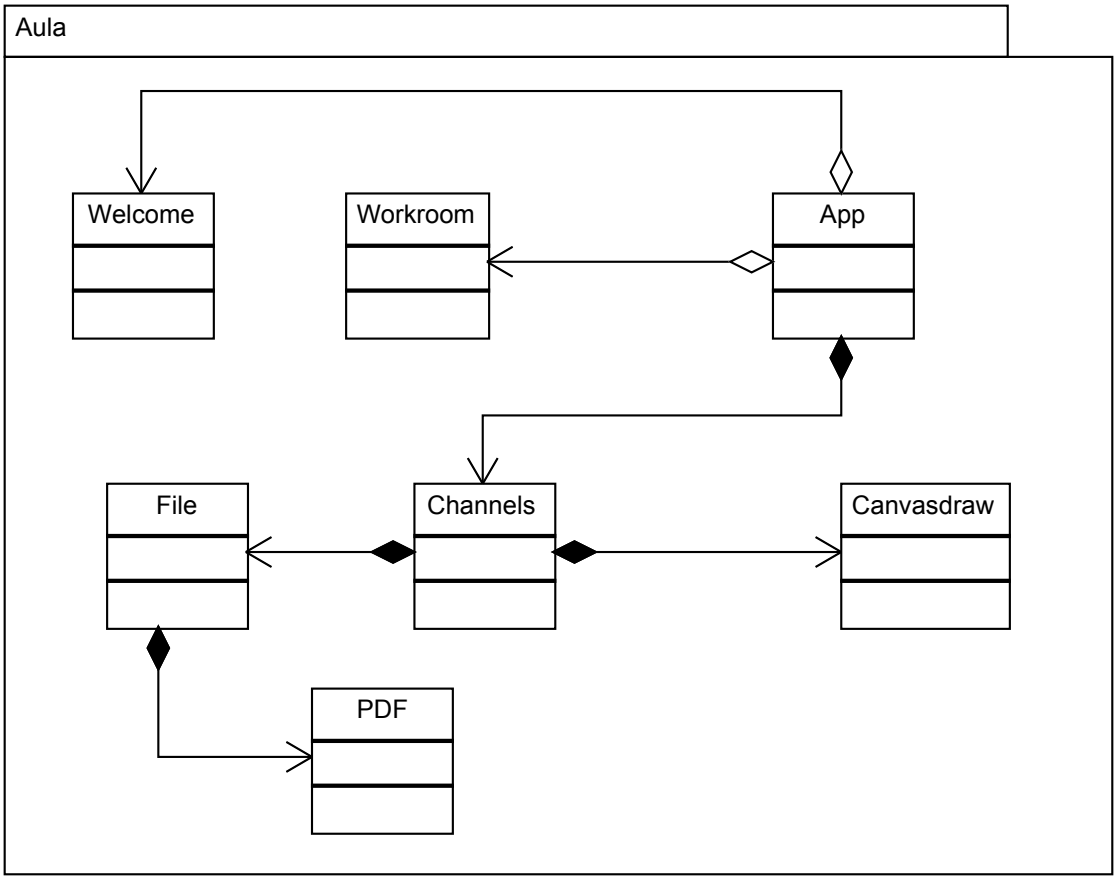

Figura 6: Diagramma classi applicazione

6.1 Welcome

Questa classe si occupa di visualizzare le prime due schermate dell'applicazione a schermo.

6.1.1 Metodi

- *initWelcome(place:string):void*
  Questo metodo renderizza la pagina iniziale di benvenuto dell'applicazione. Riceve come parametro il luogo dove visualizzare e non restituisce nessun parametro.

- *connectedWelcome():void*
  Questo metodo renderizza la pagina contenente le informazioni relative alle stanze presenti. Non riceve e non restituisce nessun parametro.

6.2 Workroom

Questa classe si occupa di renderizzare la pagina principale dell'applicazione, quella che contiene il canvas e i bottoni. Al suo intenro ci sono i metodi che permettono il modificarsi della pagina secondo il bisogno dell'utente.

6.2.1 Metodi

- *initworkroom(place:string,type:string):void*
  Si occupa di eliminare tutto ciò che è presente nella pagina per inserire ciò che ora serve all'utente. Al suo interno richiama altri metodi della classe specifici di costruire elementi necessari alla pagina. Riceve come parametri il posto dove visualizzare il tutto (place) e la tipologia di visualizzazione richiesta, se quella del professore o quella dello studente (type). Non restituisce nessun parametro.

- *buttonVideo(type:string):string*
  In base alla tipologia di utente ritorna il contenitore con i bottoni con le funzioni di inviare il video o confermare la presenza. Riceve come parametro la tipologia e restituisce una stringa html.

- *presenti():string*
  Ritorna il contenitore con la zona adibita alla visualizzaizone degli utenti presenti.

- *topRoom():string*
  Questo metodo costruisce e ritorna il widget che segnala la connessione e fornisce i bottonu di logout e exit.

- *chatRoom(place:string):void*
  Questo metodo riceve come parametro il posto dove visualizzare l'oggetto creato. Crea il widget contenente la chat, i video e i bottoni delle funzioni base.

- *video():string*
  Questo metodo costruisce la zona dove verranno visualizzati i video della connessione p2p. Restituisce un contenitore html.

- *goCanvas():void*
  Questo metodo costruisce il contenitore per il canvas e il menù sottostante. Questo metodo viene richiamato solo nel caso che la tipologia d'utente sia professore.

- *studentCanvas():void*
  Questo metodo costruisce il contenitore per il canvas e visualizza un messaggio al posto del menù perchè, essendo richiamato solo se la tipologia d'utente è studente, l'utente non è abilitato alla modifica del canvas.

- *upgrade():void*
  Questo metodo trasforma la pagina da visualizzazione studente a visualizzazione professore.

- *downgrade():void*
  Questo metodo trasforma la pagina da visualizzazione professore a visualizzazione studente.

6.3 app

Questa classe trasforma le interazioni dell'utente in azioni eseguite sui dati.

6.3.1 Metodi

- *init():void*
  Questo metodo inizializza tutto il necessario per il corretto funzionamento del programma, compreso jWebSocket. Non restituisce nulla.

- *login():void*
  Questo metono viene richiamato alla pressione del tasto login. Controlla se è stato inserito un nome utente. In caso affermativo avvia la procedura di connessione al server jWebSocket, altriemnti avvisa con un messaggio d'errore. Non ritorna nessun tipo di parametro.

- *logout():void*
  Questo metodo viene richiamato alla pressione del tasto logout. Avvia la procedura di disconnessione dal server.

- *add():void*
  Questo metodo viene richiamato alla pressione del tasto aggiungi canale. Avvia la procedura di aggiunta canale.

- *remove():void*
  Questo metodo viene richiamato alla pressione del tasto elimina canale. Avvia la procedura di eliminazione del canale.

- *enter():void*
  Questo metodo viene richiamato alla pressione del tasto della stanza dove si vuole entrare. Modifica la pagina chiedendo se si vuole accedere come professore o come studente.

- *enterWith(type:string,channel:string):void*
  Questo metodo viene richiamato alla pressione del tasto professore/studente. Riceve come parametri la tipologia di ingresso e il canale a cui si vuole accedere. Avvi ala procedura di accesso e autenticazione al canale con i diritti selezionati.

- *exit():void*
  Quando si preme il pulsante di uscita viene richiamato questo metodo che avvi ala procedura di disconnessione dalla stanza su cui eravamo e ci mostra di nuovo la visuale di scelta stanza.

- *chat():void*
  Questo metodo è richiamato alla pressione del tasto send della chat. Avvia la procedura di spedizione del messaggio scritto sulla label.

- *addS():void*
  Questo metodo viene richiamato al click di invia video per il professore o conferma presenza per lo studente. Cattura lo stream locale.

- *removeS():void*
  Il metodo è richiamato solo se l'utente è uno studente. Bisogna cliccare sul bottone abbandona posto per attivarlo. Stoppa la cattura dello stream locale e azzera timer.

- *removeStream():void*
  Solo nel caso l'utente sia un professore è possibile richiamare questo metodo. Rimuove lo streaming corrente ed elimina tutte le connessioni p2p esistenti.

- *handsUp():void*
  Metodo richiamato alla pressione del tasto hands up o hands down. Avanza richiesta di poter alzare la mano o di volerla abbassare.

- *upgrade(id:string):void*
  Metodo richiamato solo se l'utente è un professore. Traforma l'utente cliccato in professore e retocede l'utente attuale da professore a studente. Riceve come parametro l'id dello studente da promuovere.

6.4 channel

Questa classe è molto importante. Gestisce completamente la gestione della comunicazione con il server jWebSocket, la trasmissione p2p dei video e la trasmissione dei comandi impartiti dal professore al canvas.

6.4.1 Metodi

- *log(stringa:string):void*
  Questo metodo consente di stampare nella console di debug del browser tutto quello che avviene durante l'esecuzione del programma.

- *initChannels():void*
  Questo metodo inizializza il client per la connessione jWebSocket e crea tre listener per la gestione dei canali. Se jWebSocket non è supportato avvisa l'utente con un messaggio d'errore.

  - *onChannelCreatedObs(event:object):void.* Questo listener intercetta l'evento di aggiunta di un canale nel server.
  - *onChannelRemoveObs(event:object):void.* Questo listener intercetta l'evento di eliminazione di un canale dal server.
  - *onChannelReceiveObs(event:object):void.* Questo listener intercetta l'evento di ricezione della lista di canali presenti nel server.

- *login()*
  Questo metodo ci autentica al server jWebSocket. Vengono settati alcuni parametri come il tempo minimo che obblighiamo il server a tenerci connessi e l'opzione per forzare la riconnessione in caso di disconnessione accidentale. Inoltre vengono creati alcuni listener molto importanti.

  - *onOpenSocket():void.* Questo listener intercetta l'evento di apertura del socket.
  - *onWelcomeSocket():void.* Questo listener intercetta l'evento di benvenuto inviato dal server.
  - *OnMessageSocket(event:object,token:object):void.* Questo listener intercetta l'evento di ricezione di un messaggio. Questo è una delle funzioni più importanti e più lunghe della classe e basilari per il funzionamento corretto del programma. Infatti qua avviene la decodifica del messaggio ricevuto e la chiamata all'azione corretta da intraprendere in baso a ciò che si è letto dal messaggio. I messaggi ricevuti si dividono in base alla categoria di appartenenza:
    * *response* A questa categoria appartengono i messaggi ricevuti alla login, alla sottoscrizione ad un canale, alla creazione di un canale e alla richiesta di ricevere la lista delle persone connesse.

        * *data* A questa categoria appartengono i messaggi di sistema (avviso di presenza, avviso di uscita dal programma e richiesta se qualcuno trasmette stream p2p), i messaggi normali della chat, i comandi del canvas e la gestione della foto di controllo inviata dallo studente.
        * *welcome* Ricezione della connessione avvenuta con il server
        * *send* Ricezione di informazioni importanti riguardanti lo stato come la richiesta di spedire il mio nome, la richiesta di sapere i nomi delle altre persone, informazioni rigurdanti l'handshake per la comunicazione p2p, l'alzata di mano da parte dello studente e il caso si upgrade dell'utente.
    - *OnCloseSocket():void*. Questo listener intercetta l'evento di chiusura del socket del client.
    - *OnReconnecting():void*. Questo listener intercetta l'evento di autoriconnessione da parte del client se per caso si è disconnesso erroneamente dal server.

- *logout():void*
  Questo metodo disconnette il client dal server jWebSocket

- *createChannel():void*
  Questo metodo crea un nuovo canale nel server.

- *removeChannel():void*
  Questo metodo elimina un canale dal server.

- *subscribechannel(channel,accesskey):void*
  Questo metodo permette la sottoscrizione al canale passato nel parametro channel e se necessario controlla che l'accsskey sia corretta per poterci accedere.

- *getChannels():void*
  Questa funzione viene utilizzata per richiedere al server quali canali sono disponibili

- *unsubscribeChannel(id:string):void*
  Questo metodo viene richiamato quando l'utente vuole disconnettersi dal canale a cui attualmente è connesso.

- *getSubscribers():void*
  Per ottenere gli utenti connessi al canale si utilizza questo metodo.

- *auth():void*
  Per poter pubblicare messaggi sul canale bisogna autenticarsi tramite questo metodo.

- *publish(message:string,selettore:string,controllore:string):void*
  Questo è un altro metodo molto importante per l'applicazione. Tramite questo vengono scambiati tutti i messaggi di qualsiasi tipo. Riceve come parametro message che contiene il messaggio che si vuole inviare, selettore e cotrollore che servono per la decodifica corretta del messaggio da parte degli altri client. Quando si invia un messaggio com questo metodo esso viene mandato in broadcast, cioè il server lo consega a tutti i client connessi.

- *setName(username:string):void*
  Serve per settare lo username del client. Riceve come parametro appunto il nome.

- *setType(type:string):void*
  Serve per settare la tipologia di utente del client. Questo dato lo riceve come paramentro.

- *sendText(receiver:string,message:object):void*
  Questo metodo serve per mandare messaggi direttamente ad un client. Solo lui lo riceverà e il suo id è passato come parametro dentro receiver. L'altro parametro passato contiene il messaggio effettivo inviato.

- *updatePresent():void*
  Questo metodo viene richiamato soltanto se il client è stato sconnesso dal server e tenta di riconnettersi. Forza l'aggiornamento della lista dei presenti alla stanza da parte di tutti i client connessi.

- *getStream():void*
  Questo metodo chiede il permesso allo user agent di ottenere lo stream audio video della webcam. Se l'utente è un professore visualizza lo stream nel box appropriato e richama i metodi per trasmettere lo stream a tutti gli altri presenti. Se l'utente è un professore viene creato un timer che ogni tot secondi prende una foto dallo stream e la trasmette tramite websocket a tutti gli altri presenti.

- *stopStream():void*
  Questo metodo libera la webcam dalla richiesta di acquisizione dello stream e se l'utente è un professore elimina qualsiasi connessione p2p instaurata.

- *stopTimer():void*
  Questo metodo viene eseguito solo se l'utente è uno studente. Ferma il timer utilizzato per spedire una foto della presenza ogni tot secondi.

- *sendVideo():void*
  Questo metodo serve per avviare la trasmissione p2p del flusso video catturato dalla webcam.

- *addVideoStream()*
  Questo metodo viene richiamato per aggiungere la trasmissione del proprio video ad una connessione p2p esistente.

- *connectPC(id:string):void*
  La base per la connessione p2p dell'applicazione. Viene passato un parametro contenente l'id dell'utente con cui si vuole creare la trasmissione p2p e questo metodo è adibito alla creazione della connessione e alla dichiarazione di alcuni listener utili per amministrare la connessione.

  - *onAddStream(event:object):void*. Questo listener intercetta l'evento di quando la connessione p2p riceve uno stream video.
  - *onConnecting(event:object):void*. Questo listener intercetta l'evento di connessione avvenuta.
  - *onOpen(event:object):void*. Questo listener intercetta l'evento di apertura della connessione p2p.
  - *onClose(event:object):void*. Questo listener intercetta l'evento di chiusura della connessione p2p.

  - *onError(event:object):void*. Questo listener intercetta l'evento di errore causato da qualcosa nella connessione.
  - *onRemoveStream(event:object):void*. Questo listener intercetta l'evento di quando l'atro utente stoppa lo stream del video.

- *stopP2PStream():void*
  Questo metodo serve per rimuovere lo stream locale da tutte le connessioni p2p instaurate.

- *endCommunication():void*
  Questo metodo chiude tutte le connessioni p2p attive.

- *requestSomeoneIsStreaming():void*
  Questo metodo serve per richiedere se qualche utente sta trasmettendo.

- *requestHandsUp():void*
  Metodo richiamabile solo se l'utente è uno studente. Serve per richiedere l'alzata di mano.

- *alertHandsDown():void*
  Metodo usato per avvisare ch ho abbassato la mano.

- *preDrow(mex:object):void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio i dati di predraw e ricevo come parametro questi.

- *draw(x:object):void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio i dati di drow e ricevo come parametro i dati di disegno.

- *clearCanvas():void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio il comando di eliminazione della pagina corrente.

- *clearCanvasAll():void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. Invio il comando di eliminazione di tutte le pagine.

- *sendSize(size:string):void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. Invio il comando di settaggio taglia del tratto e la grandezza la ricevo come parametro.

- *sendImage(image:object):void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio il comando di caricamento immagine e la relativa immagine passata per parametro.

- *doUndo():void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio il comando di undo.

- *doRedo():void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio il comando di redo.

- *stop():void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio il comando di stop disegno.

- *sendColor(color:string):void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. Invio il comando di settaggio colore del tratto e il colore lo ricevo come parametro.

- *pagPrec():void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio il comando di andare alla pagina precedente.

- *pagSucc():void*
  Metodo usato per inviare a tutti un'azione compiuta nel canvas. In questo caso invio il comando di andare alla pagina sucessiva.

- *requestPage():void*
  Questo metodo viene usato per richiedere agli altri utenti connessi se è gia stato diseganto qualcosa sul canvas.

- *pdf():void*
  Questo metodo viene richiamato quando il professore carica un pdf per prendere l'immagine del canvas e inviarla anche agli altri utenti connessi.

6.5 file

Questa classe si occupa di interfacciarsi con la libreria esterna pdf.js adibita al caricamento dei file pdf. Tramite questa classe leggo il file pdf dal file system, lo do in pasto alla libreria che mi restituisce un metodo per visualizzarlo sul canvas.

6.5.1 Metodi

- *init():void*
  Questo metodo inizializza un listener attaccandolo al bottone del caricamento dei file

- *loadfile(evt:object):void*
  Scelto il file tramite la classica interfaccia windows viene richiamato questo metodo che come parametro riceve le informazioni relative al file da caricare. Viene creato un oggetto FileReader per leggere il file pdf nel formato corretto per passarlo alla libreria pdf.js tramite l'oggetto PDFJS. avvenuto il caricamento si procede con il richiamare un altro metodo della classe file per visualizzare la pagina.

- *renderPage(num:string):void*
  Questo metodo si occupa di renderizzare la pagina con numero passato da parametro. Il risultato viene visualizzato nel canvas.

- *localPage():void*
  Questo metrodo restituisce un oggetto contenente il numero di pagina attualmente renderizzato e il numero di pagine totali del file pdf.

- *nextPage():void*
  Questo metodo serve per visualizzare la pagina sucessiva del pdf richiamando il metodo precedente renderPage.

6.6 canvasDraw

Questa classe si occupa di disegnare sul canvas e di tutte le operazioni necessarie al corretto uso dello stesso. Questa classe era stata inizialmente sviluppata dall'altro stagista presente in azienda. Per integrarla nell'applicazione corrente si è dovuto riorgannizzare i metodi presenti e aggiungerne di nuovi per permettere la condivisione del tratto tra più utenti. Dato che la riorganizzazione ha portato la modifica di alcuni metodi è stato deciso di aggiungere features nuove per arricchire l'esperienza d'uso.

6.6.1 Metodi

- *init():void*
  Questo metodo inizializza la classe assegnando i listener delle azioni ai bottoni corrispondenti.

- *move(i:string,x:string,y:string):void*
  Questo metodo realizza la reale operazione di scrittura sul canvas del tratto. Riceve tre parametri: l'indice dell'array contenete informazioni riguardanti la linea e i punti iniziali e le coordinate x e y dei punti finali necessari per disegnare il tratto.

- *moveOther(i:string,x:string,y:string):void*
  Questo metodo viene richiamato se un altro utente ha disegnato sul canvas e l'utente corrente ha ricevuto il comando di disegnare. Riceve tre parametri: l'indice dell'array contenete informazioni riguardanti la linea e i punti iniziali e le coordinate x e y dei punti finali necessari per disegnare il tratto.

- *unDoLine():void*
  Questo metodo permette di cancellare solo l'ultimo tratto disegnato in caso di errori.

- *reDo():void*
  Questo metodo permette di ridisegnare il tratto eliminato daal'azione undo.

- *copyArrayPro():void*
  Metodo usato per la copa dell'array contenente tutti i tratti su vettori di appoggio per le azioni di redo e undo.

- *copyArray():void*
  Metodo usato per la copa dell'array contenente tutti i tratti su vettori di appoggio per le azioni di redo e undo.

- *anotherCopyArray():void*
  Metodo usato per la copa dell'array contenente tutti i tratti su vettori di appoggio per le azioni di redo e undo.

- *preDraw(event:object):void*
  Questo metodo viene invocato all'evento touchstart e contiene le operazioni per preparare il canvas per le operazioni di disegno. Riceve come parametro le coordinate del tocco.

- *preDrawM(event:object):void*
  Questo metodo viene invocato all'evento mousedown e contiene le operazioni per preparare il canvas per le operazioni di disegno. Riceve come parametro le coordinate del click.

- *preDrawOther():void*
  Questo metodo compie le operazioni di preparare il canvas per le operazioni di disegno quando riceve il comando tramite jWebSocket.
- *draw(event:object):void*
  Questo metodo inizia a scrivere la linea sul canvas quando io mi muovo con il dito sul touchscreen. Riceve come parametro le coordinate del tocco.
- *drawM(event:object):void*
  Questo metodo inizia a scrivere la linea sul canvas quando io mi muovo con il mouse sullo schermo. Riceve come parametro le coordinate del click.
- *drawOther():void*
  Questo metodo inizia a scrivere la linea sul canvas quandos riceve il comando tramite jWebSocket.
- *stopDrawM():void*
  Metodo lanciato all'evento mouseup. Stoppa l'azione di scrittura.
- *stopOtherDraw():void*
  Metodo richiamato quando il socket riceve comunicazione che chi stava disegnando ha terminato il propio disegno.
- *clearCanvas():void*
  Metodo richiamato per pulire la visuale corrente del canvas da qualsiasi tipo di disegno.
- *clearOtherCanvas():void*
  Metodo richiamato quando tramite websocket mi viene comunicato che è stato cancellato il canvas.
- *clearCanvasAll():void*
  Questo metodo è lo stesso usato sia quando l'utente clicca sul bottone che quando webSocket riceve l'istruzione corrispondente. Elimina tutte le pagine fino ad adesso create e presenta una pagina completamente bianca all'utente.
- *saveCanvas():void*
  Questo metodo apre una nuova tab con l'immagine del canvas in formto PNG pronta per essere salvata su disco.
- *selectTool():void*
  Questo metodo seleziona la tipologia di scrittura sul canvas. É possibile selezionare penna o gomma.
- *changeColor():void*
  Questo metodo cambia colore al tratto in base al bottoncino cliccato.
- *setColor(color:string):void*
  Questo metodo viene richiamato quando il websocket riceme il messaggio corrispondente e modifica il colore anche agli altri utenti. Riceve come parametro il colore nuovo da impostare.
- *changeSize(size:string):void*
  Questo metodo viene richiamato sia cliccando sul bottone con la misura del tratto voluta che qundo riceve comunicazione tramite socket. Setta la grandezza del tratto disegnato.

- *loadNewImg(file:object):void*
  Questo metodo riceve come parametro un'immagine da caricare sul canvas. L'immagine viene automaticamente ridimensionata alle dimensioni del canvas.

- *loadOtherImg(file:object):void*
  Quando un utente carica un'immagine la invia anche agli utenti i quali richiamano questa funzione per caricare a loro volta l'immagine passatagli da parametro.

- *precPage():void*
  Questo metodo si occupa di salvare il contenuto della pagina corrente e visualizzare il contenuto della pagina precedente se esite.

- *sucPage():void*
  Questo metodo si occupa di salvare il contenuto della pagina corrente e visualizzare il contenuto della pagina sucessiva se esiste altriemnti presenta un nuovo foglio bianco.

- *retPages():void*
  Quando un utente entra nella stanza può essere che ci siano già informazioni disegnate. Questo metodo invia per jwebsocket le informazioni di tutte le pagine già disegnate.

- *drawPrecedent():void*
  Appena effettuato l'accesso alla stanza, se erano gia presenti tratti disegnati sul canvas, viene richiamato questo metodo che disegna tutto quello che gli altri utenti avevano gia scritto.

7 Compatibilità

L'applicazione è compatibile sia con pc che con tablet ma con qualche limitazione. Per poter utilizzare tutte le funzionalità bisogna usare Chrome abilitando il flag che permette lo streaming p2p. É stato notato che attualmente la trasmittione p2p va a buon fine solo se i browser hanno la stessa versione.

8 Strumenti

In questa sezione verranno elencati gli strumenti hardware e software che si intendono utilizzare nella realizzazione dell'aula Virtuale.

8.1 Hardware

- Personal Computer con S.O. Microsoft Windows 7

8.2 Software

- TexStudio (per la realizzazione della documentazione in LaTeX)
- Notepad++ v5.9.6.2 (per la realizzazione di codice javascript, HTML5 e CSS)