ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

INGENIERÍA DE TELECOMUNICACIÓN
+
INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

PROYECTO FIN DE CARRERA

Diseño e implementación de un cliente de videomensajería y simulación de procesamiento Cloud

Autor: Julio Iván Alegre Torrejón
Tutor: Gregorio Robles Martínez

Curso académico: 2012/2013

Proyecto Fin de Carrera

Diseño e implementación de un cliente de videomensajería y simulación de procesamiento Cloud

Autor: Julio Iván Alegre Torrejón
Tutor: Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 2013, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2013

"Elige un trabajo que ames y no tendrás que trabajar un sólo día de tu vida"
Confucio

Agradecimientos

En primer lugar me gustaría agradecer a Telefónica I+D la oportunidad que me ha dado al trabajar con ellos. No sólo colaborar en proyectos europeos como este, sino además la oportunidad de presentarlo como mi proyecto de fin de carrera. Gracias también a Gregorio por tener siempre un hueco para solucionarme los problemas que he podido tener al redactar esta memoria.

A mis padres, por la educación y el apoyo que me han dado a lo largo de estos casi 24 años, no se qué estaría haciendo ahora mismo sin ellos. A mi hermana por soportarme siempre sea como sea y quererme tanto. En general a toda mi familia por hacerme sentir tan especial y concretamente a mi abuela Estrella. Sé que estés donde estés, estarás tan orgullosa de mí tanto como todos lo estamos de ti.

A todos los amigos que he podido tener desde pequeño, porqué aunque no todos me han aportado felicidad, sí que me han hecho convertirme en lo que soy. A Josu, Jesús, Sonia, Isma, Revi por darme una vía de escape cuando no podía mas.

A toda esa gente que me ha acompañado desde que llegué a la universidad. Desde el primer día en el que me senté al lado de una tal Nazareth y fui conociendo a Luis, Javi, Carlos, Alba y un Adrián muy especial. Ya más adelante, grandes momentos de Bertypartys, montaditos, pizzas y fiestas con Berty, Isra, Fran, Guille, Poker, Elena, Abel...

Por último a toda la gente de mi Erasmus, que aunque el contacto se vaya perdiendo, nunca volveré a ser igual y no es sólo culpa mía.

Resumen

El objetivo principal del proyecto es la realización de un prototipo completo de cliente de videomensajería que use la tecnología *Signspeak*, que convierte lengua de signos a texto, y una infraestructura cloud en la que poder ejecutar varias instancias de dicha tecnología.

El cliente de mensajería está formado por cuatro componentes bien diferenciados. Cada componente tiene su función y está desarrollado de forma independiente. Todos ellos se conectan entre sí a través de HTTP y usan XML cuando hace falta enviar datos estructurados.

El primer componente es una aplicación web realizada con Sencha Touch. A este componente se le llamará de ahora en adelante **Frontend**. En este *Frontend* se pueden leer mensajes y ver vídeos adjuntos a estos. En la parte derecha de la pantalla tenemos la lista de mensajes entrantes, y en la izquierda una visualización completa del mensaje. La segunda función del *frontend* es crear mensajes con vídeo. Esta pantalla es un modal en el cual podremos adjuntar vídeos previamente guardados mediante *Drag and Drop*.

El segundo componente es **Video Uploader**. Es una aplicación de escritorio escrita en .NET con la cual se granan vídeos desde la webcam, se convierten a un formato compatible para la web y se envían junto con una captura al **Backend**.

El **Backend** provee a la solución persistencia y gestión avanzada. Es un servidor programado en Java para Tomcat y Jersey. Su función es guardar notas y vídeos y servirlos cuando haga falta.

La **infraestructura Cloud** son un conjunto de servidores en Tomcat que se encargan de distribuir peticiones entre ellos para la transcripción de vídeos. Pero puede ser modificada para usarse en cualquier otro entorno de distribución de trabajo.

Índice General

Índice de Figuras

Índice de Tablas

Acrónimos

EUD	**E**uropean **U**nion of the **D**eaf
REST	**RE**presentational **S**tate **T**ransfer
HTTP	**H**ypher**T**ext **T**ransfer **P**rotocol
AJAX	**A**synchronous **J**avascript **A**nd **X**ML

Parte I

Introducción y objetivos

Capítulo 1

Introducción

En el presente capítulo se pretende dar una visión global del proyecto. Para ello se introduce el contexto en el que se ha desarrollado el trabajo realizado. En segundo lugar se explica la tecnología base sobre la que funciona el software. Por último se define el estado del arte de las herramientas desarrolladas.

1.1. Contexto del PFC

El presente proyecto de fin de carrera se corresponde con el trabajo realizado en Telefónica I+D dentro del proyecto europeo Signspeak [1]. Dicho trabajo ha sido realizado dentro de una beca de 20 horas semanales bajo la supervisión de Maria del Carmen Rodríguez Gancedo (mcrg@tid.es). En el proyecto también he colaborado en el proyecto con Francisco Javier Caminero Gil (fjcg@tid.es) y Álvaro Hernández Trapote (alvaro.trapote@gmail.com). El proyecto se desarrolló entre Noviembre de 2011 y Abril de 2012. Una vez concluido el proyecto, me prorrogaron la beca y me reasignaron a otro proyecto.

En uno de los entregables del proyecto, se definieron varias aplicaciones y casos de uso para la tecnología Signspeak. En esta memoria se detalla el desarrollo e implementación de un prototipo para uno de estos casos de uso: Un cliente de videomensajería. El escenario que aparecía en el entregable era el siguiente:

> Mary quiere enviar un correo a varias personas. Algunos de ellos pueden oír mientras que otros no. Ella graba un vídeo en lengua de signos y lo envía. La tecnología Signspeak lo convierte a texto y lo envía por email con el vídeo y su transcripción a todas las direcciones.

Telefónica eligió este caso de uso para su posterior diseño e implementación. De esta forma tenemos un escenario y objetivo claro: desarrollar una aplicación que permita intercambiar mensajes con vídeo y que transcriba dichos vídeos usando Signspeak. Éste es el punto de partida de todo el posterior desarrollo.

A dicha aplicación se le asignó el nombre de **VideoSL Mail** (Video Sign Language Mail)

Una vez finalizado el desarrollo e implementación del software previsto, la aplicación fue evaluada por la EUD [2] con resultados satisfactorios.

Cabe destacar que el proyecto Signspeak está enmarcado en el FP7 (Seventh Framework Programme) [3] que subvenciona proyectos de investigación a todo tipo de empresas y entes públicos.

1.2. ¿Qué es Signspeak?

Signspeak es un proyecto europeo cuyo objetivo es el desarrollo de una tecnología capaz de convertir lengua de signos a texto.

La comunidad sorda prefiere usar el lengua de signos para comunicarse, ya que es su forma natural de comunicación. Esto significa que para cualquier persona sorda, le es mucho más natural, fluido y cómodo hablar en lengua de signos que hablar de forma escrita. De hecho, según la Federación de Personas Sordas de Cataluña (FESOCA), el 90 % de los sordos son analfabetos funcionales [4].

Con Signspeak, personas sordas o con problemas auditivos que hablen lengua de signos, puede comunicarse sin problemas con personas no sordas sin que se requiera de ellos que hablen lengua de signos. Además, se elimina una de las barreras mas importantes para la comunidad sorda: la comunicación electrónica. Signspeak abre un abanico inmenso de posibles soluciones y servicios tecnológicos que hasta entonces eran imposibles.

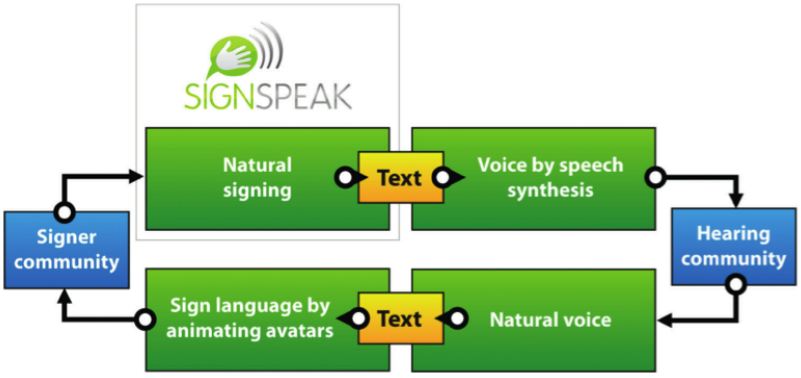

Figura 1.1: Esquema de integración de Signspeak con otras tecnologías

Como podemos ver en la Figura 1.1 Signspeak es el complemento perfecto para una comunicación entre la comunidad sorda y oyente perfecta. De esta forma y con un complemento de texto a voz (text-to-speech), una persona sorda podría convertir cierto mensaje en lengua de signos y hacérselo llegar a cualquier persona oyente. Esta persona lo escucharía en su lenguaje natural. Por otro lado, a través de avatares y motores de voz a texto (speech-to-text), conseguiríamos que una persona sorda pudiera recibir un mensaje de una persona oyente sin que éste tenga que saber lengua de signos.

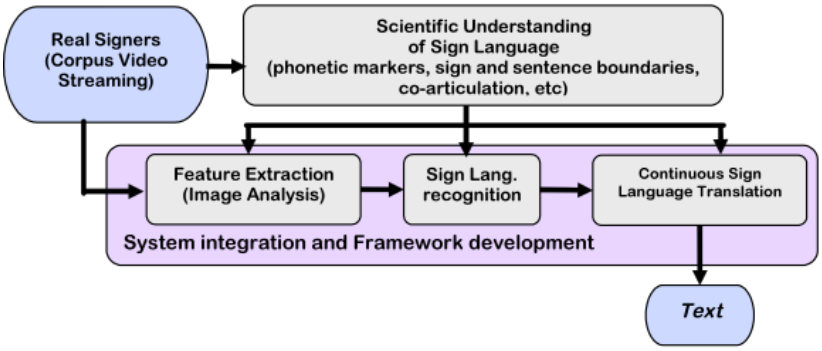

Figura 1.2: Procesos y componentes de Signspeak

Los componentes de la tecnología Signspeak se pueden ver en la Figura 1.2. Como entrada tenemos un vídeo de una persona hablando en lengua de signos. La tecnología está compuesta en primera instancia de una extracción de características a través de análisis de imagen. El siguiente componente provee de reconocimiento de lengua de signos a través de las características extraídas en el componente anterior. Y por último una traducción a texto del lengua de signos reconocido a través del vídeo.

1.3. Estado del arte

En este apartado explicaremos diversas alternativas a determinadas partes del software desarrollado. Para esta sección, hemos dividido la aplicación en dos partes:

1. **Cliente de videomensajería**: Que, en nuestra solución se compone de Interfaz web, VideoUploader y Backend
2. **Infraestructura Cloud Computing**

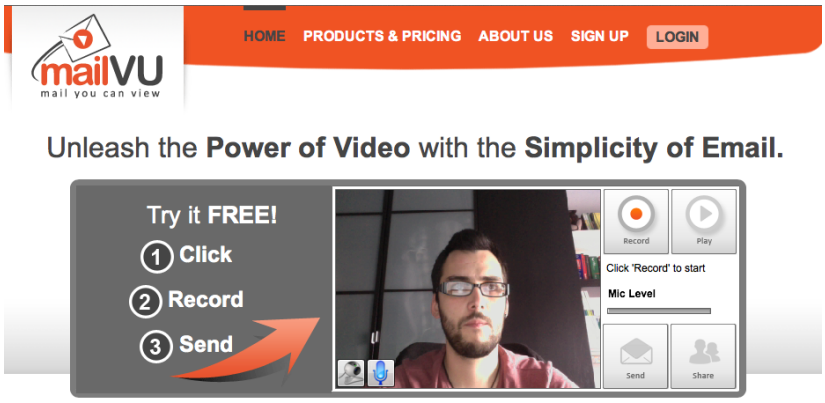

Figura 1.3: Captura del sitio web **mailVU**

1.3.1. Cliente de videomensajería

Dentro de los clientes de videomensajería podemos distinguir varios tipos. Para empezar todos aquellos servicios que sirvan para enviar mensajes de vídeo al uso desde un ordenador estándar. Cabe destacar que en todos ellos, la reproducción de vídeo se hace a través del plugin Adobe Flash. Dicha dependencia ocasiona en muchos casos la incompatibilidad con ciertos dispositivos móviles. En este apartado podemos encontrar **mailVU** o **eyejot**

mailVU[5]: Proporciona videomensajería a través de su aplicación web y aplicaciones nativas para iOS y Android. En la Figura 1.3 podemos ver una captura de la prueba gratuita que ofrece el servicio. Tiene tres tipos de cuentas para poder usarse, todas ellas de pago. En los planes más avanzados se puede incluso cambiar la plantilla de la interfaz para adaptarla a tu empresa. Por último, dispone de una API para que diferentes aplicaciones puedan subir vídeos a través de su servicio.

eyejot[6]: Servicio de videomensajería a través de aplicación web y aplicación nativa para iOS, pero no Android. El uso de la aplicación online es gratuito y la aplicación para iOS tiene un precio de 3.99$. Se pueden recibir notificaciones de mensajes entrantes al correo electrónico, en los cuales se puede ver el vídeo directamente desde el gestor de correo. En la Figura 1.4 podemos ver una captura de la aplicación web.

Con la explosión de los smartphones, han proliferado infinidad de aplicaciones móviles de mensajería con las cuales se puede añadir vídeo. Estas aplicaciones no son aplicaciones de videomensajería, pero para determinados casos podrían valer. La mayoría funcionan únicamente a través del dispositivo móvil (como **Whatsapp** [7] o **Spotbros** [8]). El caso de **Line** [9] es especial, ya que tiene versión de escritorio, pero desde esta plataforma no permite añadir vídeo grabado en tiempo real.

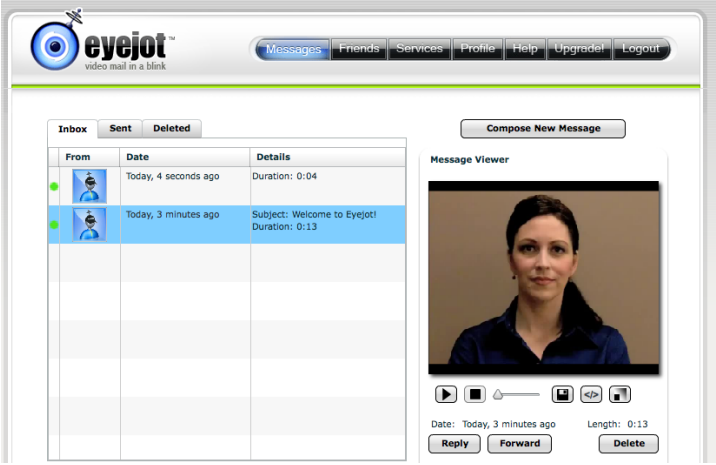

Figura 1.4: Captura de la aplicación web **eyejot**

Además de los mencionados clientes de videomensajería y las aplicaciones móviles con opción de adjuntar vídeo, existen multitud de aplicaciones y servicios de videochat. Entre ellos el más conocido sea quizás **Skype** [10]. También se podría destacar **Google Hangout** [11], que permite videochat con múltiples usuarios a la vez.

Una vez vistas varias alternativas de videomensajería podemos observar cómo todas estas soluciones son soluciones comerciales finales. Ninguna de ellas permite la transcripción de lengua de signos a través del vídeo grabado. Tampoco provee ninguna solución la capacidad de poder añadir la tecnología Signspeak a su servicio.

1.3.2. Cloud computing

Cloud computing o computación en la nube, es un modelo de provisión de recursos computacionales a través de internet. De esta forma se delega el almacenamiento, procesado, abastecimiento y persistencia de los datos en otra empresa.

Se analizarán las diferentes soluciones Cloud computing a través de su clasficación. A saber: Software as a Service, Platform as a Service e Infraestructure as a Service.

Software as a Service (SaaS):

Este es un modelo orientado a usuarios finales, para usar, no para desarrollar. La necesidad es un servicio directo; el usuario no requiere de una máquina virtual o un entorno de programación. Algunas soluciones SaaS son:

Google Apps [12]: Solución con la que cualquier empresa puede ofrecer servicios de correo electrónico, suite ofimática, calendario, contactos y demás bajo

su propio dominio. Todos los servicios disponen de interfaz web por lo que son multiplataforma. Son soluciones finales en las que se pueden personalizar ciertos detalles.

Office 365 [13]: Similar a Google Apps pero centrado en productos Microsoft. Incluye Exchange para correo y calendario, Lync para chat, SharePoint para compartir archivos y las aplicaciones web de la suite Microsoft Office. Todas estas aplicaciones están mucho más presentes en las empresas que las de Google, pero por contra sus versiones web están menos maduras que las de su principal competidor.

Platform as a Service (PaaS):

Modelo intermedio en el que únicamente se desarrolla el software requerido sin necesidad de tener en cuenta una planificación de recursos. El proveedor ofrece una infraestructura sobre la que desplegar tu aplicación. Cuando necesite más recursos, el proveedor, por sí solo, se encargará de redimensionarse para satisfacer las necesidades de la aplicación. Con una solución PaaS podemos obtener funcionalidad completa para nuestro software sin tener en cuenta la complejidad que conlleva la administración y el escalado de recursos. Algunas soluciones PaaS son:

Google App Engine [14]: Tendremos que desarrollar la aplicación desde un inicio usando su framework. Una vez hecho se despliega en los servidores de Google, y éste se encarga de añadir o quitar recursos disponibles según sea necesario. Si la plataforma es suficiente para satisfacer las necesidades de tu aplicación es la solución más fácil si empiezas un proyecto de cero. Está pensado para funcionar con Python y Java, aunque soporta más lenguajes. La facturación se calcula dependiendo de el número de peticiones recibidas, aunque dispone de un intervalo gratuito. Quizás sea la más conocida dentro de las soluciones PaaS.

RedHat Openshift [15]: Solución mucho más flexible que Google App Engine. Permite la instalación de tus propias aplicaciones Python (con Django o Flask), Node.js, Ruby on Rails, PHP (Zend y CodeIgniter) y Java (JBoss, Spring y Tomcat). Tiene planes gratuitos y de pago. Tal como Google App Engine te permite la redimensión automática, pero con más control. Mucho más orientado a empresas que han desarrollado su aplicación en frameworks o tecnologías ya maduras y únicamente quieren desplegarlas sin tener que tener en cuenta el redimensionado de recursos. Dispone de un plan gratuito con funcionalidad completa y de un plan de pago (se empieza a cobrar una vez superado el límite gratuito), que incluye más recursos, soporte y certificados avanzados.

Infraestructure as a Service (IaaS):

Modelo de más bajo nivel en el que no sólo se ha de desarrollar el software, sino que también se gestionan los recursos. Con un modelo IaaS se tienen acceso a máquinas virtuales completas con administración total y gestión completa no sólo del número de máquinas virtuales, sino de los recursos de los que dispondrá cada una. Bajo cualquiera de las siguientes soluciones tenemos máquinas con gestión total. En cualquiera de las máquinas se puede instalar el sistema operativo que se quiera y configurar sin límite.

Amazon EC2 [16]: Quizás la solución IaaS más conocida. Nos proporciona instancias de máquinas virtuales en cuestión de segundos, pudiendo modificar los recursos asociados a dichas máquinas. Dispone de una API para crear y modificar dichas instancias. Se integra fácilmente con el servicio de almacenamiento Cloud de Amazon S3. Provee también de una función de autoescalado.

Windows Azure [17]: Alternativa de Microsoft con imágenes para máquinas virtuales ya preparadas para ejecutar software en .NET, Node.js, Java y Python.

Otras alternativas IaaS en las que no se ha profundizado son Arsys Cloud Computing, BT, 1&1 o Ubuntu Cloud.

Más adelante se explicará la solución Cloud propuesta y cómo se integra (si es que lo hace) con las soluciones estudiadas.

1.4. Estructura de la memoria

La presente memoria se ha estructurado partes, cada una de las cuales se divide en capítulos.

- **Parte I, Introducción y objetivos:** Antes de explicar el trabajo realizado, se introducen algunas ideas y conceptos relacionados con el proyecto. En el primer capítulo se detalla el contexto en el que se desarrolló el trabajo realizado. Por otro lado se explica la tecnología sobre la que se basan las aplicaciones desarrolladas. Además, se explican alternativas al proyecto realizado o posibles infraestructuras sobre las que desplegar dicho proyecto. En el segundo capítulo se indica la meta del proyecto y los objetivos que se deben cumplir.
- **Parte II, Diseño e implementación:** En esta parte se explicará la arquitectura del software desarrollado, explicando cada aplicación por separado y cómo interactúan entre ellas. Por otro lado, profundizando un poco más se indicarán errores y problemas que surgieron a la hora de implementar y ampliar el software.

- **Parte III, Discusión y conclusiones:** Primero se comentarán las evaluaciones de la EUD al software desarrollado. Por último, terminará la memoria valorando si se han cumplido los objetivos y profundizando en líneas futuras.

Capítulo 2

Objetivos

La meta principal del proyecto es:

> Crear un software de videomensajería en la nube que use la plataforma Signspeak.

Dicha meta la podemos desglosar en los siguientes objetivos:

1. **Diseñar una arquitectura**:

   Debemos decidir qué componentes tendrá el software a desarrollar. Analizar las funcionalidades necesarias y decidir las aplicaciones necesarias para que la solución cumpla las expectativas.

   Por ejemplo, para el caso del cliente de videomensajería, debemos decidir si se grabará vídeo directamente desde el navegador usando flash o alguna otra tecnología, o por contra, se requerirá de una aplicación de escritorio.

   Se decidirá si se usará una solución estándar para el procesado distribuido de vídeo (una de las soluciones del estado del arte definido en 4.6.1) o si se usa una solución propia.

2. **Decidir la tecnología sobre la que se desarrollará cada aplicación**

   Una vez tengamos claros los componentes que necesita nuestra aplicación, decidimos qué tecnologías debe usar cada componente. Se considerará si ya hay algún tipo de software desarrollado en poder del equipo de trabajo para aliviar carga.

3. **Diseñar las interacciones entre componentes**

Una vez definidos los componentes y aplicaciones que hay que desarrollar, se define cómo interactuarán estas. Qué tipo de comunicación se usará o qué formato de datos se usará para la transmisión.

4. **Desarrollar cada componente**

   En un inicio de forma desacoplada, para poder probar y depurar de forma mucho más fácil. Una vez desarrollado cada componente se irán conectando entre sí y comprobando que funciona antes de proseguir con la integración.

5. **Evaluación por parte de la EUD**

   Una vez desarrollado todo y teniendo una solución estable, se envían unidades de prueba a la Unión Europea de Sordos (EUD). Allí será evaluado por personas sordas de las cuales se obtendrán *bugs* y posibles mejoras.

Parte II

Diseño e implementación

Capítulo 3

Arquitectura y tecnologías usadas

En este capítulo se explicarán las decisiones iniciales de arquitectura, tecnologías que se tomaron al realizar el proyecto y por último una arquitectura final.

En un primer lugar, se detallará la división en componentes que se realizó. Dichos componentes deben ser desarrollados en un inicio de forma aislada para poder probarse fuera de la arquitectura completa. Una vez probados de forma aislada, se irán conectando componentes a la arquitectura final hasta ver que funcione todo de forma global.

Por otro lado se explicarán las tecnologías que usa cada componente y la decisión de porqué se han usado dichas tecnologías.

Además se indicarán los métodos y especificaciones de la comunicación entre componentes, además de los modelos de datos para almacenarlos y tratarlos.

Por último se indicarán los dispositivos usados para el desarrollo y las pruebas.

3.1. Arquitectura inicial

En la Figura 3.1 podemos observar la arquitectura global del cliente de videomensajería integrado en una infraestructura Cloud. Tenemos dos partes bien diferenciadas: **Client side** y **Server side**.

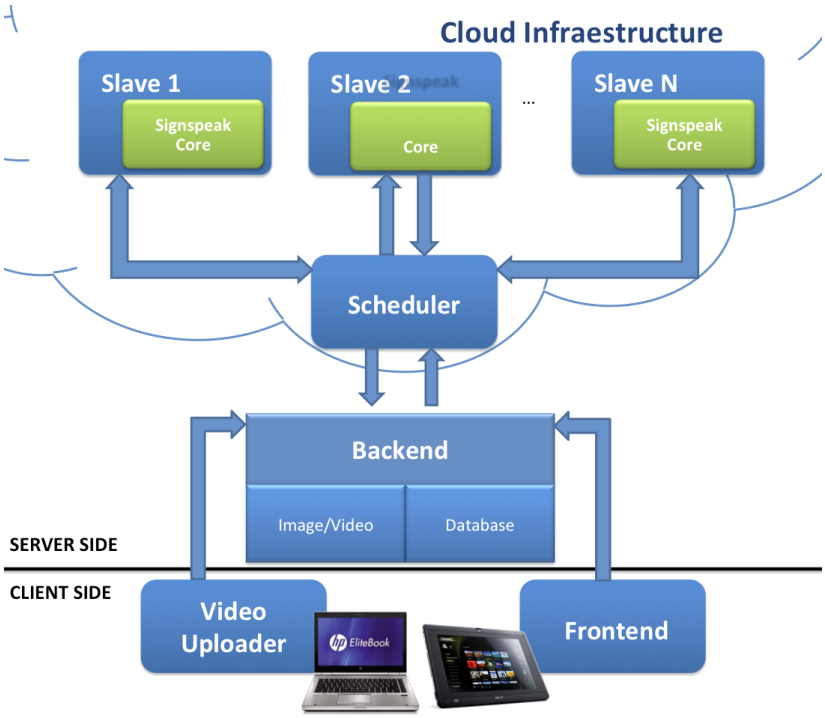

Figura 3.1: Arquitectura general de VideoSL Mail

3.1.1. Client side

Esta parte se compone de todo el software con el que interactuará el usuario final. Es decir, que en la parte del Cliente van todas las aplicaciones que puedan ser utilizadas por el usuario.

En esta parte nos encontramos con dos componentes. El **Video Uploader** se encargará de la grabación de vídeo y el **Frontend** será el encargado de visualizar y gestionar mensajes.

3.1.1.1. Video Uploader

Video Uploader será una aplicación de escritorio que grabará vídeo a través de la webcam del usuario, hará una captura en cierto punto de ese vídeo y lo enviará todo al **Backend**.

Esta aplicación es la que más desalineada está con el resto de componentes. Por ello se trató de minimizar el trabajo a realizar. Para tal propósito se buscaron alternativas de software libre ya desarrollado para posteriormente añadirle la funcionalidad que faltara.

Una vez analizadas algunas soluciones, se decidió partir como base de la aplicación simple-webcam-recorder [18]. Una aplicación sencilla escrita en .NET para grabar vídeos sin audio usando la librería AForge.NET [19].

Analizando los objetivos que necesitamos para nuestra aplicación encontramos:

- Tener una pantalla de autenticación.
- Tener un interfaz personalizado para grabar vídeos.
- Grabar vídeo desde una cámara web.
- Tener dicho vídeo en un formato compatible con la mayoría de navegadores web.
- Capturar una imagen del vídeo para que pueda ser usada como previsualización.
- Enviar el vídeo, la imagen y el usuario al backend.

El software sobre el que se parte, ya nos proporciona la grabación de vídeo. El vídeo grabado se almacena en un archivo *avi* con vídeo descomprimido. Por otro lado dispone de filtros y opciones que no sirven para nuestra aplicación.

Por un lado debemos adaptar la interfaz para que sea lo más simple posible. Debemos proveer de una pantalla inicial en la que se pida usuario y contraseña para identificar quién graba el vídeo. En la pantalla principal deberíamos hacer que la previsualización abarque lo máximo posible de ventana. Por otro lado las únicas interacciones posibles del usuario deberían ser grabar vídeo y parar vídeo.

La librería multimedia AForge.NET sobre la que se basa simple-webcam-recorder sólo provee como formato de salida *avi* descomprimido y *wmv*. Ninguno de estos formatos es compatible con los navegadores modernos para ser visualizados sin necesidad de plugins (como Flash). Debido a esta limitación, nos vemos en la necesidad de recodificar el vídeo grabado para que ocupe el mínimo espacio posible y minimizar la carga del servidor. Para ello se ha usado FFmpeg [20].

FFmpeg nos permite convertir vídeos desde línea de comando, por lo que puede ser ejecutado desde multitud de tecnologías y lenguajes de programación. Una vez se haya grabado el vídeo con la librería AForge.NET, se recodificará a H.264 MPEG-4 AVC (en adelante MP4) usando FFmpeg.

Se ha decidido usar MP4 en vez de Theora o WebM ya que al ser más antiguo las librerías están mas avanzadas. Por otro lado dispone de un buen soporte tanto en navegadores de escritorio como navegadores móviles [21].

Por otro lado, para ciertos aspectos de la reproducción de vídeo y cosas que serán necesarias en el frontend, hará falta realizar una captura del vídeo. Dicha captura se hará desde el código del Video Uploader, sin ejecutar programas o funcionalidad externa.

3.1.1.2. Frontend

Uno de los pilares base de la solución a desarrollar. El frontend será la ventana de interacción principal para el usuario final. Será una aplicación web que dispondrá de las siguientes funcionalidades:

- Vista general de mensajes entrantes con extracto del texto contenido y avatar del remitente.
- Vista detallada del mensaje seleccionado incluyendo visualización de vídeo sin plugin a través HTML5.
- Composición de mensajes a través de un modal. Tendrá campos de texto y un carrusel de vídeos subidos por el usuario. El vídeo que se quiera añadir se adjuntará al soltar su previsualización sobre una zona de arrastre.
- Debe permitir ver vídeos subidos de una forma fácil y rápida.

De esta forma, el usuario podrá ver una lista de los mensajes entrantes, ver el contenido de dichos mensajes (incluyendo vídeo) y escribir nuevos mensajes pudiendo incluir vídeos previamente guardados con la herramienta Video Uploader.

De esta forma podemos observar como se ha intentado simplificar al máximo la herramienta de grabación de vídeo, para que recaiga casi todo el peso de la gestión y creación de mensajes en el frontend. Al ser una aplicación web, en principio se podría acceder desde varios dispositivos o sistemas operativos.

3.1.2. Server Side

En la parte del servidor (ver Figura 3.1), podemos distinguir dos componentes bien diferenciados: **Backend** y **Cloud Infraestructure**.

Esta parte será completamente invisible al usuario final. No deberá tener acceso ni al código ni a la gestión de estos componentes. En los componentes del lado del servidor se proveerá a la solución de persistencia, interconexión entre componentes y procesado en la nube.

3.1.2.1. Backend

El backend será el enlace natural entre toda la parte del cliente (**Video Uploader** y **Frontend**) y la parte servidor. Se encargará de:

- Almacenar mensajes, usuarios y demás datos en una base de datos.
- Almacenar y servir imágenes y vídeos como servidor de archivos.
- Proveer de una API para crear, modificar y borrar mensajes.
- Mandar a la **Cloud Infraestructure** transcribir los vídeos necesarios proveyendo alguna URL de retorno en la que se guarde dicha transcripción.

El Backend es el pilar fundamental de la parte del servidor. Guardará datos textuales y binarios en base de datos y archivos. A través de una API REST toda la funcionalidad será accesible desde fuera. De esta forma el Video Uploader enviará el vídeo y su captura encapsulado como paquetes HTTP.

Así se simplifican y estandarizan las interconexiones entre componentes. En todas las tecnologías es mucho más fácil implementar peticiones HTTP que usar sockets o cualquier otra alternativa.

Por otro lado, se encargará de que, cuando reciba un vídeo, pedir a la Cloud Infraestructure una transcripción de dicho vídeo. El Backend también deberá proveer a la Cloud Infraestructure una forma de enviarle la trancripción de forma asíncrona. Sin necesidad de bloquear el hilo de ejecución.

3.1.2.2. Cloud Infrestructure

Ésta será una infraestructura desarrollada para distribuir la carga de transcripción de vídeos con la tecnología Signspeak. Estará compuesto de una sola máquina que distribuirá la carga (*Scheduler*), y de tantas máquinas ejecutando Signspeak como se quiera (*Slaves*).

Las funcionalidades que debe desarrollar con eficacia este componente son:

- Número de máquinas *Slave* flexible
- Protocolo de descubrimiento de máquinas *Slave* por parte del *Scheduler*
- Distribución de peticiones de transcripción entre *Slaves*
- Devolución de las transcripciones conseguidas al Backend.

Para empezar, debemos tener en cuenta que esto está planteado para que se puedan añadir tantas máquinas para procesar como se quieran. De esta forma cuando una máquina disponible para procesar se "da de alta", el Scheduler debe ser notificado de ello para tenerlo en cuenta en sus futuras distribuciones de trabajo.

Para ello, se supondrá que cada máquina Slave conoce a la máquina Scheduler. De esta forma, cuando una máquina Slave pasa a estar activa, envía un mensaje "Hello" al Scheduler.

La distribución de trabajo se realiza comprobando el uso de CPU instantáneo. Debe ser tenido en cuenta a la hora de elegir tecnología. Cuando se requiera de realizar una tarea, el Scheduler pedirá a cada Slave su uso de CPU, una vez obtenido, le asignará la tarea al Slave que menos cargado esté (según este uso de CPU).

Estas peticiones de trabajo serán bloqueantes, por lo que el Scheduler se bloqueará hasta conseguir respuesta del Slave correspondiente. Una vez tenga la transcripción, se la enviará al Backend.

3.2. Tecnologías usadas

En la Figura 3.2 podemos ver qué tecnologías se han usado en cada componente del software desarrollado. Por otro lado, se identifican todas las interconexiones entre componentes como comunicaciones usando una arquitectura RESTful.

3.2.1. Video Uploader

Se barajaron distintas posibilidades entre las APIs de HTML5 para el manejo de vídeo. Pero eran todavía demasiado primitivas para ser usadas en un navegador estándar de escritorio. En concreto HTML Media Capture [22] se ajustaba bastante bien a la funcionalidad requerida (grabar vídeo en un formato amigable para la web desde una cámara). Pero este API no funcionaba en navegadores de escritorio.

HTML Media Capture funciona a la perfección en dispositivos Android. Su uso es realmente simple, se añade un elemento *file* de formulario con ciertos atributos modificados. En el navegador por defecto en versiones de Android superiores a 3.0, al pulsar dicho elemento de formulario, lanza directamente la videocámara. Graba el vídeo que se quiera en vivo con una aplicación nativa y lo almacena en un archivo. Por último, el navegador inyecta este fichero como el archivo de entrada del elemento de formulario. En HTML5Rocks se detalla su uso de una forma exquisita [23].

Lamentablemente casi todo el software estaba enfocado para un usuario de PC de sobremesa de forma que HTML Media Capture podría quedar como alternativa, pero no nos es suficiente para cumplir toda la funcionalidad requerida. Para PC había que desarrollar un software más integrado en toda la plataforma.

Como ya se explicó en la sección anterior, este componente se aleja un poco de la funcionalidad principal de VideoSL Mail, por lo que se ha tratado de minimizar

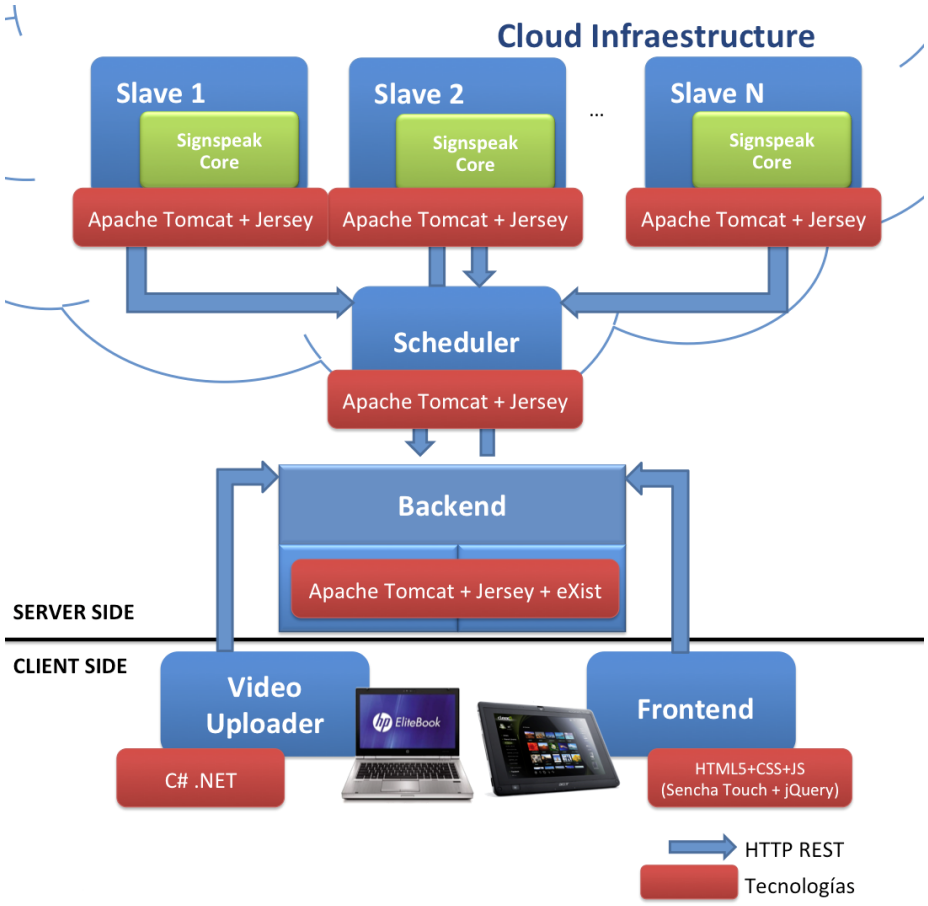

Figura 3.2: Tecnologías usadas en cada componente de VideoSL Mail

el trabajo usando alguna herramienta existente como base. Como Simple Webcam Recorder [18] está escrito en C# para .NET, esta ha sido la tecnología usada para grabar vídeos desde la webcam.

Video Uploader se conectará con el Backend usando peticiones HTTP (REST), lo que facilita su implementación para las interconexiones entre diferentes tecnologías y lenguajes de programación.

3.2.2. Frontend

Desde un primer momento se supuso que el Frontend sería web. Por lo que la elección de HTML5 + CSS3 + JavaScript respecto de cualquier otra alternativa estaba clara. De esta forma quedaba sólo la elección de si se usaba algún *framework* para facilitar el diseño o si por contra se hacía todo desde cero.

Debido a la poca experiencia desarrollando software para la web, se decidió que sería mucho más fácil usar algún framework web para aliviar gran parte del diseño CSS3.

El equipo decidió que lo mejor para desarrollar la aplicación web sería usar Sencha Touch [24], ya que era compatible con la mayor parte de los navegadores móviles y Chrome y Safari para ordenador de escritorio. Era un framework bastante más maduro que otras alternativas como jQuery Mobile [25] (en aquel momento carecía de versión estable).

El uso de Sencha Touch no limitaba en modo alguno poder usar cualquier API de HTML5 o el uso de cualquier otra librería. Debido a diversos problemas que se explicarán en el siguiente capítulo, se añadió la librería JavaScript jQuery para la recepción y manejo de eventos. No era necesario, pero de nuevo facilitó bastante el desarrollo de ciertas partes de la aplicación web.

Por último, todas las conexiones entre Frontend y Backend son peticiones HTTP (REST) a través de AJAX. El Frontend no tiene porqué estar alojado en la misma máquina que el Backend. De esta forma se podría modularizar para convertir en una aplicación nativa para diferentes sistemas operativos.

3.2.3. Backend

En este caso ya se habían realizado previamente en el equipo servidores escritos en Java para Apache Tomcat [26], con Jersey [27] para añadir la interfaz REST y base de datos XML con eXist [28]. De esta forma ya se tenían bases y código realizado que, aunque no fuera muy útil a la hora de reutilizar, si que fue muy útil como código de ejemplo.

3.2.3.1. Java Servlets y Jersey

Ya tenía conocimientos previos acerca de los servlets de Java, pero no su uso en conjunción con Jersey. Los servlets de Java son una tecnología muy madura ya, ampliamente documentada y con amplio soporte en la mayoría de servidores.

Jersey le añade a los servlets de Java la capacidad de administrar verbos REST de forma fácil y cómoda. De esta forma podemos definir métodos para determinadas urls y extraer parámetros de forma relativamente automática.

3.2.3.2. eXist

eXist es un sistema de bases de datos basadas en XML en la que no existen esquemas ni es necesaria la definición de un modelo inicial de base de datos, lo que aporta gran flexibilidad.

Las consultas a la base de datos se hacen a través de XQuery, un lenguaje de consulta del W3C para consultas en bases de datos XML. Bastante potente, ya que por un lado es similar a SQL, pero te permite otros tipos de consultas más parecidas a la navegación por carpetas.

Con una base de datos XML conseguimos una simplicidad espectacular a la hora de rellenar datos iniciales, modificar datos manualmente sin hacer uso de XQuery o a la hora de exportar la base de datos.

3.2.4. Cloud Infraestructure

Aquí de nuevo, para reutilizar la mayor parte de las configuraciones se ha decidido usar también servidores en Java para Apache Tomcat con Jersey.

Debido a que estos componentes no necesitan persistencia, ni Scheduler ni Slaves necesitan de ninguna base de datos XML (eXist).

3.3. Arquitectura final

Ya hemos visto primero la división en componentes del proyecto. Después se han detallado las tecnologías con las que se trabajará en cada componente, y los protocolos para su interconexión (REST).

En la arquitectura final se explicarán los verbos HTTP que se usarán para las comunicaciones. Se detallarán las URLs sobre las que accederá y servirá cada pieza de software.

El esquema general se puede ver en la Figura 3.3. Como se indica en la leyenda, en mayúsculas se indica el verbo HTTP usado y en minúsculas, el nombre del recurso al que se accede para realizar determinada tarea con sus parámetros entre paréntesis.

3.3.1. Video Uploader

Recapitulando, el Video Uploader grabará vídeo desde una videocámara, hará una captura de dicho vídeo y lo enviará junto al usuario al que pertenece dicha captura al Backend. En este apartado se explicará la interconexión entre el Video Uploader y el Backend.

De esta forma, una vez grabado el vídeo ya disponemos de toda la información necesaria para enviar al Backend. En la Tabla 3.1 podemos ver el formato de la trama que se envía cada vez que se sube un vídeo.

Los elementos entre corchetes son parámetros variables. De esta forma vemos como el usuario se envía como parámetro, al igual que lo hace el nombre del archivo.

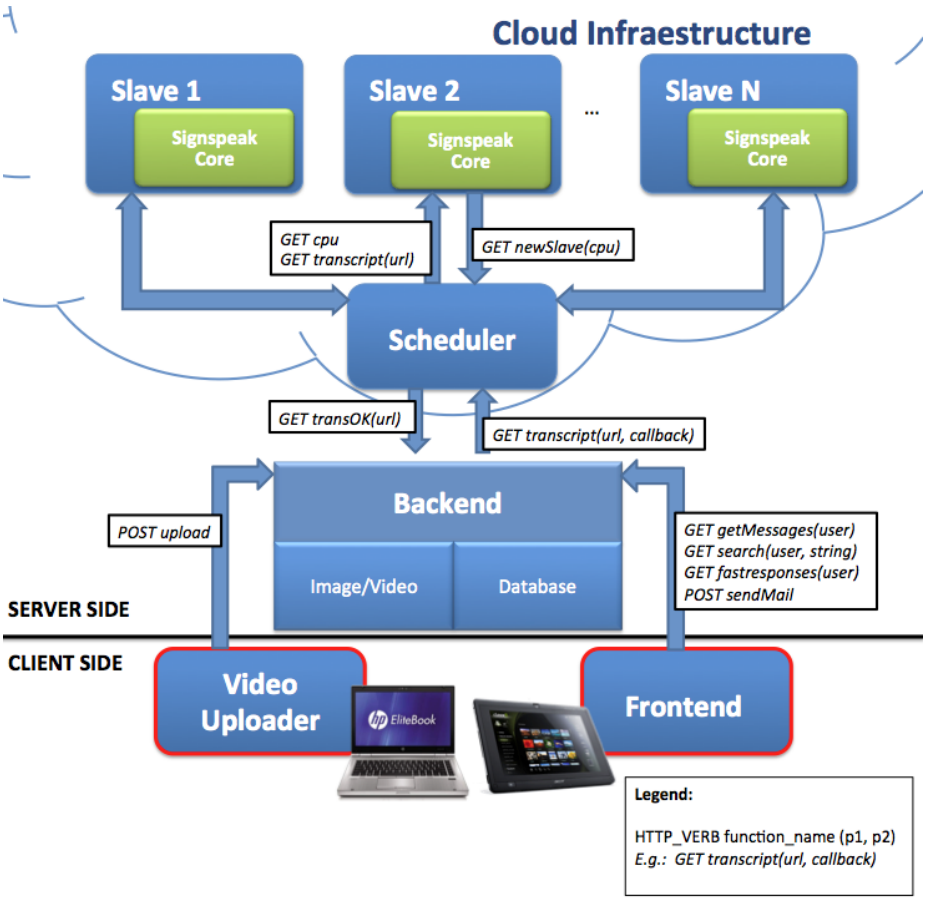

Figura 3.3: Arquitectura final de Video SL Mail

Content-Disposition:form-data; user="[user]"; name="file"; filename="[filename].mp4"; Content-Type: video/mp4 –video– [Bytes of the converted video] –video– [Bytes of the preview image]

Tabla 3.1: Formato de mensaje POST *upload*

Tenemos un separador entre cabecera y datos binarios que es –*video*–. Este separador nos sirve para separar los bytes de dos archivos diferentes como son el vídeo grabado y la previsualización capturada.

3.3.2. Frontend

Antes de definir ninguna interconexión ni descripción de comunicación con cualquier otro componente. Hay que definir una estructura para el almacenaje y procesado de mensajes.

3.3.2.1. Formato de mensajes XML

Teniendo en cuenta que todo quedará almacenado en una base de datos XML, lo lógico será transmitir información usando XML. De esta forma el Backend, en muchos casos devolverá directamente el resultado de las *queries* que haga a la base de datos.

El XML que deberá procesar el Frontend al arrancar incialmente sigue el siguiente esquema:

Listing 3.1: Lista de mensajes de VideoSL Mail

```
<?xml version="1.0" ?>
<signspeak_mail>
       <message>
          <from>frommail@signspeak.eu</from>
          <to>
                 <adressee type="recipient">[first.recipient]
                    @signspeak.eu</adressee>
                 <adressee type="cc">[carboncopy.recipient]
                    @signspeak.eu</adressee>
          </to>
          <labels>
                 <label name="[Label1]" color="#FF0000"/>
          </labels>
          <tags>
                 <starred/>
                 <important/>
                 <unread/>
          </tags>
          <subject>[Subject]</subject>
          <body/>
          <videos>
                 <video_transcription filename="[videoUrl].mp4"
                    thumbnail="[thumbnailUrl].png">
```

```
                    [Transcription]
                </video_transcription>
            </videos>
        </message>
        <message></message>
</signspeak_mail>
```

De esta forma tenemos un elemento raíz *signspeak_mail*, y entro de él irán tantos elementos *message* como se quieran. Tendrá destinatarios que podrán ser del tipo *recipient* (directos) o *cc* (carbon copy). Cada mensaje podrá tener etiquetas a las que se les puede asignar un color. Los *tags* son indicadores del mensaje, pudiendo estar marcado como favorito (*starred*), marcado como importente (*important*) o como no leído (*unread*).

Un mensaje dispondrá de un asunto (*subject*), podrá disponer de un mensaje en texto (*body*) y de un vídeo con una transcripción asociada (*videos*). Como podemos observar, el vídeo tiene una url del fichero asociado, una url a la previsualización del vídeo y el elemento contiene la transcripción del mismo.

3.3.2.2. Estructuras internas JavaScript

Las comunicaciones en XML simplifican enormemente las interconexiones entre componentes. El problema es que es mucho más cómodo usar una estructura interna a la hora de acceder recurrentemente a datos almacenados en memoria. Por eso, aunque el Frontend reciba XML, lo procesará y almacenará la información en listas y objetos internos.

Hay dos listas principales que almacenarán las información visualizada en la aplicación web:

- **DataMsg**: Almacenará la información necesaria de todos los mensajes del usuario para que puedan ser representados con todo su contenido. Es decir tendrán toda la información requerida para la vista detallada del mensaje.

  Sigue el siguiente formato:

  Listing 3.2: Informacion detallada de mensajes

```
[{
                subject: "[Subject]",
                from: "[from]@signspeak.eu",
                date: "[Hour, Day]",
                starred : [true or false],
                important: [true or false],
```

```
                unread: [true or false],
                labels: [
                        {
                                name: '[Label1]',
                                color: "#[FF0087]"}
                        ],
                to: [
                        '[first.recipient]@signspeak.eu'
                ],
                cc: [
                        '[carboncopy.recipient]@signspeak.eu'
                ],
                body: "[Body of the message]",
                video: [{
                        filename: '[videoUrl].mp4',
                        thumbnail: '[thumbnailUrl].bmp',
                        transcription: '[Transcription]'
                }]
        }
]
```

NOTA: En el fragmento mostrado sólo se detalla un elemento, habrá tantos elementos en la lista como mensajes del usuario.

- **MsgsList**: Almacenará la información necesaria de los mensajes para ser mostrados en la lista de mensajes. Sigue el siguiente formato:

Listing 3.3: Informacion básica de mensajes

```
[
        {
                profileImg: "[userThumbnail].png",
                name: '[first.recipient]@signspeak.eu',
                hour: "[Hour]",
                subject: '[Subject]',
                starred : [true or false],
                important: [true or false],
                unread: [true or false],
                extract: [First 40 characters of body],
                preview: '<img src="[thumbnailUrl].bmp" width
                    ="93" height="52" />',
                labels: [
```

```
                        {
                                name: '[Label1]',
                                color: "#[FF0087]"
                        }
                ]
        }
]
```

NOTA: En el fragmento mostrado sólo se detalla un elemento, habrá tantos elementos en la lista como mensajes del usuario.

3.3.2.3. Comunicaciones del Frontend

Una vez definido el modelo con el que tratarán Frontend y Backend a la hora de procesar y almacenar los datos, se explicarán las comunicaciones entre ambos.

- **GET getMessages(user)**: Petición por la que el Frontend, una vez autenticado, pide al Backend una lista completa de todos los mensajes de ese usuario (*user*). El Backend devolverá algo equivalente a Listing 3.1
- **GET search(user, string)**: El usuario podrá buscar una cadena de texto dentro de todos sus mensajes, mostrando los que tengan alguna coincidencia. De nuevo devolverá una lista de mensajes que seguirá el esquema de Listing 3.1
- **GET fastresponses(user)**: Este método devuelve una lista en formato JSON de los vídeos subidos al Backend por ese usuario. El formato de la respuesta es:

Listing 3.4: Lista de vídeos subidos con Video Uploader

```
[
        {
                img: '[thumbnailUrl].bmp',
                title: '[Video Title]',
                date: '[Date of the video]'
        }
]
```

Esta lista contendrá tantos elementos como vídeos haya subido el usuario. La URL del vídeo vendrá denotada por la URL de la previsualización, pero con extensión *mp4*.

- **POST sendMail**: Este será el verbo HTTP por el cual se enviarán los mensajes que se creen en el Frontend. El formato de estos mensajes será exactamente igual al de cualquier elemento *message* de Listing 3.1.

3.3.3. Backend

Ya se han descrito las comunicaciones entre las dos aplicaciones con las que interactuará el usuario y el Backend. Definiremos en esta sección las comunicaciones entre Backend y Cloud Infraestructure.

Sólo dos son los métodos a través de los cuales se comunican Backend y Cloud Infraestructure: *GET transcript(url, callback)* y *GET transOK(url)*.

El método *transcript* se lanzaría cuando, desde el Frontend se haya enviado un mensaje. De este modo debemos obtener usando la tecnología Signspeak, una transcripción del vídeo subido a través de Video Uploader. La localización del vídeo vendrá dada por el parámetro *url*. El parámetro *callback* designará la URL en la que vendrá dada la respuesta a la transcripción.

Se supone que la transcripción de un vídeo no es una tarea trivial, de forma que esta comunicación no será bloqueante, es decir, se enviará una petición de transcripción, y cuando la Cloud Infraestructure haya terminado de procesarla, será esta misma la que le hará una petición HTTP a la URL definida en el *callback*.

Para nuestro caso, hemos definido que el *callback* siempre será *transOK*. De esta forma, cuando la Cloud Infraestructure termine de procesar y transcribir el vídeo, hará una petición web al Backend al método *transOK* con la URL del vídeo como único parámetro.

3.3.4. Cloud Infraestructure

Antes de profundizar en las comunicaciones de este componente, cabe recordar que sólo existe un Scheduler y puede haber tantos Slaves como se quieran. Los Slaves notificarán al Scheduler que han arrancado para que sean tenidos en cuenta a la hora de planificar tareas y ya estará disponibles para realizar lo que le pida el Scheduler.

El Scheduler es la cabeza visible de toda la Cloud Infraestructure, es el único que tiene comunicación con el exterior, el único que recibe peticiones del Backend. Cuando el Scheduler reciba una petición de transcripción, le enviará a cada Slave conectado que envíen su uso de CPU actual. El que menos cargado esté será el encargado de transcribir el vídeo usando Signspeak.

Así, analizadas las comunicaciones Video Uploader-Backend, Frontend-Backend y Backend-Cloud Infraestructure, únicamente quedan por analizar las comunicaciones internas dentro de la Cloud Infraestructure. Estas son:

- **GET newSlave(cpu)**: Este método será llamado siempre que un Slave arranque. Hará una petición al Scheduler enviándole su uso actual de CPU.
- **GET cpu**: Método por el cual el Scheduler pide a cada Slave su uso de CPU.
- **GET transcript(url)**: Una vez que el Scheduler haya decidido a qué Slave le encarga la transcripción del vídeo, le hará una petición usando este método pasándole la URL del vídeo a transcribir.

De esta forma, recapitulando lo visto en esta sección: el usuario se autentica y graba un vídeo con Video Uploader. Este vídeo se sube al Backend a través del método *POST upload*. El usuario se autentica en el Frontend, al arrancar, la aplicación obtiene los mensajes del Backend a través de *GET getMessages(user)*. El usuario podría buscar mensajes (*GET search(user, string)*).

Por otro lado, al lanzar una composición de mensajes, se cargarían los vídeos subidos previamente a través de *GET fastresponses(user)*. Si el vídeo finalmente se envía, el Frontend enviaría un mensaje *POST sendMail* a través del cual el Backend recibiría, guardaría y si fuera necesario, pediría a la Cloud Infraestructure una transcripción para el vídeo recientemente enviado.

3.4. Dispositivos utilizados

En esta sección se presentarán los terminales que se han usado para el desarrollo, pruebas y evaluación del software VideoSL Mail. Hace falta tener en cuenta que estos dispositivos eran los que se tenían antes de planificar cualquier arquitectura. De esta forma, la arquitectura se debía adaptar a los dispositivos existentes, no al revés.

3.4.1. Portátil HP EliteBook 8470p

Fue el portátil de la iniciativa que se me asignó al realizar el proyecto (ver Figura 3.4). Con este ordenador se realizaron todos los desarrollos. En él se ejecuta el Backend, se aloja el Frontend, y se tienen al menos 3 máquinas virtuales VirtualBox. Una de estas máquinas virtuales es el Scheduler de la Cloud Infraestructure y las otras dos son Slaves de la misma.

Por otro lado, alojaba una copia de los binarios de Video Uploder. Pero, como no disponía de webcam, se requirió de una cámara web externa para la grabación de vídeos.

El sistema operativo huésped es Windows 7 y el sistema operativo de las máquinas virtuales es Ubuntu 11.10.

Figura 3.4: Portátil usado para el desarrollo y parte de las pruebas

3.4.2. Tablet Acer Iconia W500

La interfaz a desarrollar era una interfaz táctil, con paneles arrastrables y botones grandes. De esta forma era necesario un dispositivo táctil de grandes dimensiones en el que pudiera cargar la interfaz web con todo su contenido. Así, con el Iconia W500 (ver Figura 3.5), se disponía de una herramienta mucho más dinámica y natural para probar el Frontend. Por otro lado, gracias a llevar Windows 7 en su interior, se pudo ejecutar sin problemas el Video Uploader.

3.4.3. Otros dispositivos

Además de los anteriores terminales, se usó un **Samsung Galaxy Nexus** para la prueba del API HTML5 MediaCapture. Se probó también el Frontend fugazmente en un **Samsung Galaxy Tab 10.1**.

Figura 3.5: Tablet usada para las pruebas del Frontend y Video Uploader

3.5. Seguridad del servicio

Debido a que todo el software desarrollado es un prototipo, no se ha considerado ningún aspecto de seguridad. De esta forma podemos observar como las interfaces REST de los componentes no usan autenticación de ningún tipo. Todos los métodos del Backend son accesibles por cualquier usuario en su misma red.

Con el Video Uploader se puede observar algo parecido, no se comprueba ninguna credencial y el vídeo se envía de forma no segura el Backend.

La finalidad del proyecto no es hacer un software siempre estable y seguro; es ver hasta dónde se puede llevar la plataforma Signspeak. Así, se ha enfocado el tiempo mucho más en desarrollar más funcionalidad y componentes que en hacer menos pero de forma segura.

Capítulo 4

Implementación del software

En el presenta capítulo se detallarán decisiones y acciones no contempladas en la arquitectura (ver Capítulo 3), además de los principales problemas con los que me he enfrentado a la hora de llevar el diseño del software y arquitectura a código fuente. En este caso, se explicará de forma cronológica. De esta forma distinguimos las siguientes etapas:

1. Desarrollo aislado e independiente del Frontend
2. Desarrollo inicial del Backend y adaptación del Frontend
3. Desarrollo de Video Uploader sin conexión con el Backend
4. Actualización del Backend y conexión de Video Uploader con éste
5. Desarrollo de Cloud Infraestructure
6. Actualización del Backend para la conexión con la Cloud Infraestructure

Así, una vez desarrollados todos los componentes, el software está listo para ser probado y usado por la EUD.

4.1. Desarrollo aislado e independiente del Frontend

El Frontend ha sido el componente que más tiempo y esfuerzo ha requerido por mi parte para la realización del proyecto. Cuando me asignaron la tarea de escribir un cliente de videomensajería usando Sencha Touch, mi experiencia con interfaces web era prácticamente nula. Únicamente había hecho pequeñas modificaciones a temas Wordpress, pero modificando solamente HTML, en ningún

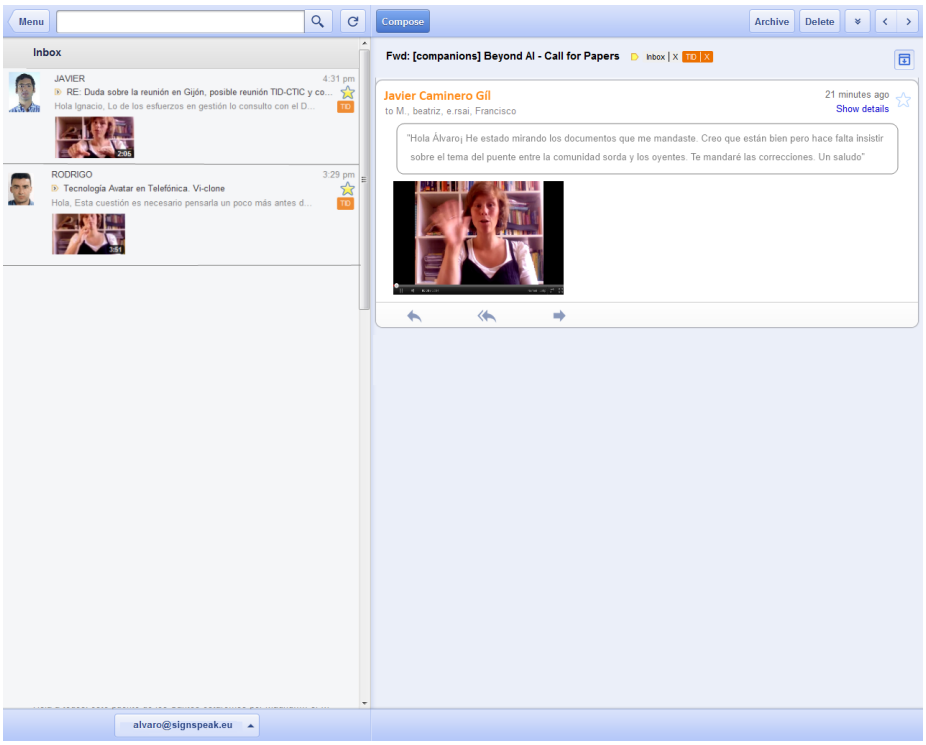

Figura 4.1: Mockup de la bandeja de entrada de VideoSL Mail

caso JavaScript. Por otro lado, en la carrera no se impartió materia alguna de JavaScript hasta la asignatura de Sistemas y Aplicaciones Telemáticas (SAT), asignatura que cursé después de haber iniciado el desarrollo del Frontend.

Todo el Frontend inicial se alojó en un servidor Apache HTTP Server local. Más adelante veremos como el Frontend pasó a formar parte del servidor Apache Tomcat 7.0 que contenía el Backend.

Al inicio, se me enviaron tres bocetos (o mockups) de los principales componentes del Frontend. En sucesivas secciones se partirá del boceto inicial y se explicará el desarrollo realizado.

4.1.1. Bandeja de entrada

Como se puede ver en la Figura 4.1, la bandeja de entrada estaría compuesta de una barra de herramientas superior e inferior y de dos paneles principales. Lo primero fue poner estas barras de herramientas usando componentes de Sencha Touch.

En Sencha Touch todos los componentes se definen desde el JavaScript (a diferencia de otros frameworks como jQuery Mobile). De esta forma, se definen objetos propios de Sencha Touch para prácticamente todo. Si un objeto tiene un

atributo llamado *xtype*, éste objeto pasa a ser un objeto Sencha Touch y viene definido por el valor de este atributo. También se pueden crear objetos a través de *new* y el tipo de objeto. Los objetos Sencha siempre empiezan por *Ext.*. Así, una barra de tareas se puede definir como

var toolbar = new Ext.Toolbar({...});

o como un objeto con el *xtype* definido como *toolbar*. Por otro lado, cada barra de tareas estaba compuesta de botones, selectores y campos de texto. Cada uno tenía una correspondencia con algún componente Sencha Touch.

Establecer barras de tareas y llenarlas de componentes Sencha fue bastante sencillo. El problema vino cuando se quisieron establecer componentes más personalizados como la lista de mensajes. Por ello se definieron las barras inferior y superior como componentes Sencha, pero los dos paneles centrales se consideraron componentes con HTML puro (*no-Sencha*).

Uso de Templates Sencha

Insertar todo este HTML en el código JavaScript no es una solución elegante, por lo que se usaron Templates Sencha Touch. Esta funcionalidad de Sencha Touch permite añadir código HTML a los ficheros .html. Este código no será interpretado por el navegador al cargar la página, pero puede ser recuperado e interpretado cuando sea necesario a través del JavaScript. Además permite la inserción de ciertos aspectos no presentes en HTML como condicionales o acceso a variables. De esta forma, en nuestro archivo html tenemos el siguiente fragmento de código:

Listing 4.1: Ejemplo de Templates Sencha Touch

```
<textarea id="listPane" class="x-hidden-display">
        <tpl for=".">
                <tpl if="unread">
                        <div class="msg unreadMsg" id="msg{#}">
                                {extract}
                        </div>
                </tpl>
                <tpl if="unread != true">
                        <div class="msg" id="msg{#}">
                                {extract}
                        </div>
                </tpl>
        </tpl>
</textarea>
```

Para poder procesar este fragmento HTML desde el JavaScript, primero se crea el panel (variable *listPane*) configurándolo para que cargue el Template *listPane*, que es el id del *textarea* contenedor de todo el código template. Una vez creado, se procesa con:

listPane.update(msgsList);

De esta forma, la línea *<tpl for="."*> itera sobre la lista *msgsList*. Dentro de cada objeto, *<tpl if="unread"*> comprueba el atributo unread; si es *true*, se devolverá el contenido del nodo. Después si unread no es *true* (no existe *else*), se mostrará el elemento HTML que contenga el *tpl if*. Nótese que se muestran los contenidos del objeto actual con llaves. Con {#} se muestra el número de iteración del bucle en el que se esté inmerso.

Hoja de estilos

Una vez declarado el HTML, era necesario definir un estilo para los nuevos componentes puros. Se empezaron a añadir propiedades a las clases e ids en un archivo CSS. De nuevo, mi experiencia en el mundo del diseño web era nula, de forma que pronto se convirtió en un pequeño caos. Se detectaron clases sin visibilidad y confusiones con algunos atributos *style* dentro del HTML. Llegó un punto en el que había aprendido bastante acerca de diseño web con CSS, de forma que rehice por completo el fichero css con los nuevos conocimientos para eliminar posibles problemas futuros.

La primera versión de la interfaz fue con HTML estático embebido en cada componente. De esta forma cada mensaje era un elemento HTML que se definía en código. Más tarde se fue haciendo todo más dinámico para ir preparándolo para la conexión con el Backend.

Escribir el HTML necesario para la visualización de cada componente central no originó muchos problemas. Las modificaciones se debían a cambios de opinión del equipo acerca de la disposición de ciertos elementos elementos como los botones de Responder, Responder a todos y Reenviar. Como elementos a destacar se añadieron bordes redondeados con CSS3 y el uso del tag video de HTML5 para la reproducción de vídeo sin necesidad de complemento por parte del navegador. En cierto momento se decidió prescindir de las etiquetas debido a la limitación de tiempo impuesta por la comisión europea.

De esta forma ya teníamos una interfaz en HTML5 completamente estática y sin funcionalidad, pero visualmente completa. Este esquema se mantendría hasta el final. Se puede ver en la Figura 4.2 el aspecto final de la interfaz web. Como se puede notar, la disposición de los dos paneles centrales está intercambiada. Esto es así por las opiniones recibidas por parte de la EUD (lo cual será explicado en un capítulo posterior).

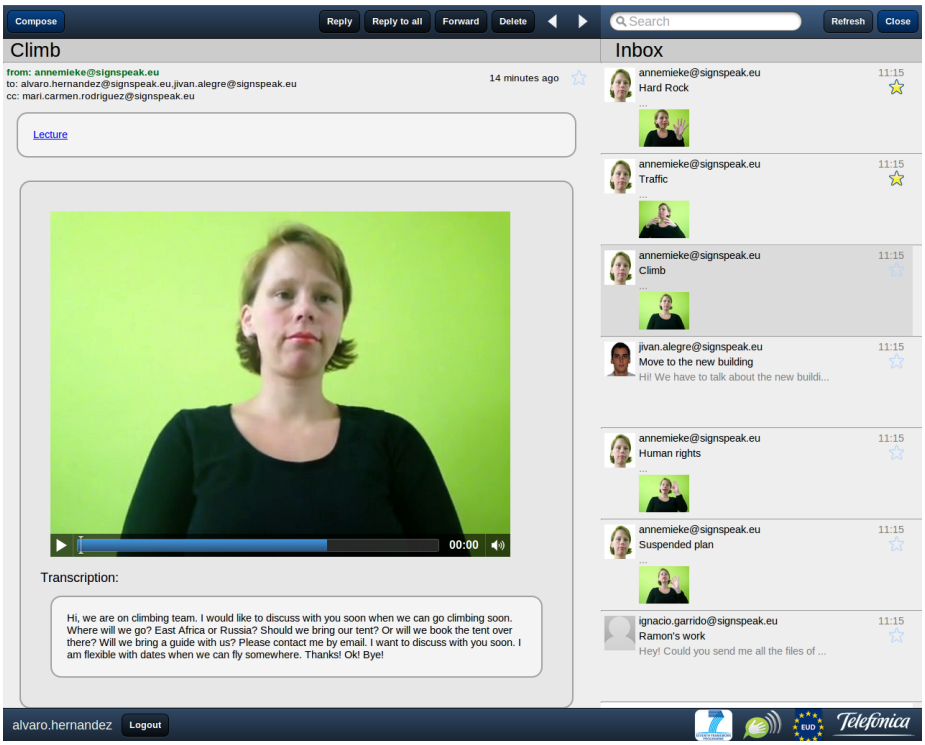

Figura 4.2: Bandeja de entrada de VideoSL Mail

Primera versión dinámica

Una vez en este punto del desarrollo, se convirtió el código HTML estático para que cargara el contenido de los mensajes a través de dos variables: *msgsList* y *dataMsg*. En un principio estas variables estarían definidas en el código JavaScript como listas, pero en el futuro, estas variables serían el resultado de procesar la respuesta del Backend.

msgsList contendría la información de los mensajes necesaria para que sea mostrada en el panel de lista de mensajes (parte central derecha de Figura 4.2). En el extracto siguiente se muestra la información de un solo mensaje, esta lista contendría tantos objetos como mensajes existan.

Listing 4.2: Información relevante para la lista de mensajes

```
[
        {
                profileImg: "[userThumbnail].png",
                name: '[first.recipient]@signspeak.eu',
                hour: "[Hour]",
                subject: '[Subject]',
                starred : [true or false],
```

```
            important: [true or false],
            unread: [true or false],
            extract: [First 40 characters of body],
            preview: '<img src="[thumbnailUrl].bmp" width="93"
               height="52" />',
            labels: [
                        {
                               name: '[Label1]',
                               color: "#[FF0087]"
                        }
                  ]
      }
]
```

Como se puede observar, existe el atributo labels, ya que en un principio estaban planificadas como funcionalidad, pero como ya se ha comentado anteriormente, se decidió prescindir de ellas. En *dataMsg* pasa lo mismo.

La lista *dataMsg* contiene la información que necesita el panel de visualización detallada del mensaje (panel central izquierdo). De nuevo, se muestra sólo un objeto, pero es una lista que contiene tantos objetos como mensajes se tengan.

Listing 4.3: Información relevante para la vista detallada de un mensaje

```
[{
            subject: "[Subject]",
            from: "[from]@signspeak.eu",
            date: "[Hour, Day]",
            starred : [true or false],
            important: [true or false],
            unread: [true or false],
            labels: [
                   {
                          name: '[Label1]',
                          color: "#[FF0087]"}
                   ],
            to: [
                   '[first.recipient]@signspeak.eu'
            ],
            cc: [
                   '[carboncopy.recipient]@signspeak.eu'
            ],
            body: "[Body of the message]",
            video: [{
```

```
                    filename: '[videoUrl].mp4',
                    thumbnail: '[thumbnailUrl].bmp',
                    transcription: '[Transcription]'
            }]
    }
]
```

De esta forma se rellenaron manualmente estas listas y se modificó el código para que en vez de mostrar HTML estático, cargara la información de estas variables. Recordemos que en este apartado no se carga ningún dato de ningún servidor, es un desarrollo aislado. Modificar el código para que mostrara la información de estas variables no resultó ser ningún problema. y se pudo pasar al siguiente hito, la recepción de eventos.

Manejo de eventos

Dentro de los componentes HTML *puros*, se empezaron a manejar varios eventos, como la selección de ciertos campos (por ejemplo, las estrellas) o la selección de mensajes en la lista de mensajes. Se intentaron definir *listeners* o receptores de eventos para esos componentes a través de Sencha Touch, pero no se lograron hacerlos funcionar. En componentes Sencha Touch no había ningún problema para invocar cierta función cuando se pulsaba sobre él (por ejemplo, botones Sencha), pero por la razón que fuera, no logré conseguir capturar eventos de elementos HTML *puros*.

Es por eso por lo que me recomendaron usar jQuery para la recepción de eventos. A partir de aquí todo fue mucho más fácil e intuitivo. Con jQuery, a través de selectores CSS se podían editar elementos o añadir receptores de eventos de una forma sencilla. De esta forma, si se seleccionaba la estrella de la vista detallada del mensaje, la estrella correspondiente en la lista de mensajes se vería modificada al instante. De la misma forma se enviaría un evento al pulsar sobre cualquier mensaje en la lista de mensajes.

La recepción de eventos se facilitó sobremanera gracias a la incursión de la librería jQuery, pero este añadido trajo consigo algún que otro problema. En concreto, algunas veces algunos receptores de eventos se desactivaban al cambiar de mensaje. De esta forma, cada vez que se pulsaba en algún mensaje de la lista de mensajes, era necesario activar receptores de eventos para los componentes de la vista detallada.

Preparando el modal de composición

De esta forma ya teníamos una bandeja de entrada con gran parte de la funcionalidad total y funcionando de forma independiente. La siguiente pantalla

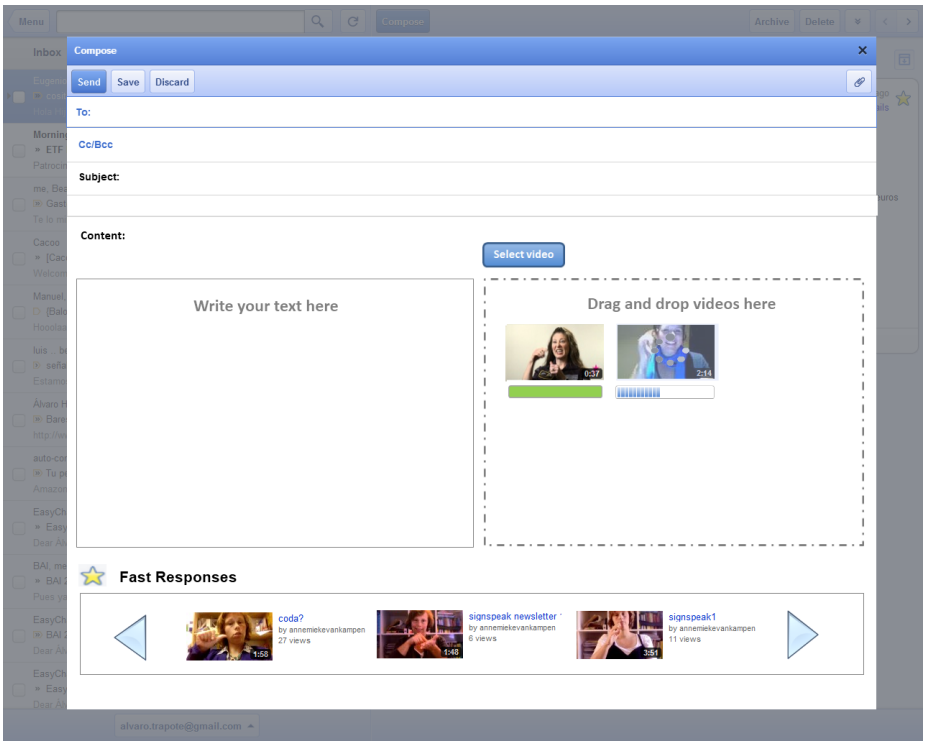

Figura 4.3: Mockup del modal dedicado a la composición de mensajes

a realizar fue el modal para la composición, que sería lanzado al pulsar sobre *Compose*, *Reply*, *Reply all* y *Forward*.

Con *Compose* se lanzaría un modal con los campos de destinatario y contenido del mensaje vacíos. Con *Reply* se añadiría al campo *to* el remitente del mensaje seleccionado, con *Reply all* se añadirían al campo *to* todos los destinatarios del mensaje seleccionado. Además en ambos casos se añadiría al contenido del mensaje a crear "Previous message: z el contenido del mensaje seleccionado. Con *Forward* simplemente se añadiría al contenido del mensaje "Forwarded message: z el contenido del mensaje seleccionado.

4.1.2. Composición de mensajes

El boceto inicial del panel modal para la composición de mensajes se puede ver en la Figura 4.3. Se pueden distinguir una barra de herramientas superior, campos de destinatarios para los mensajes, un campo de asunto, un campo de contenido y dos zonas de arrastre. Una de ellas es un carrusel de previsualizaciones de vídeo y otra zona en la que arrastrar dichas previsualizaciones.

En la Figura 4.4, se puede ver el resultado final de la interfaz desarrollada. Añadir la barra de tareas superior fue igual de fácil que con la bandeja de entrada, se le añadieron todos los botones del mockup menos el de añadir un archivo

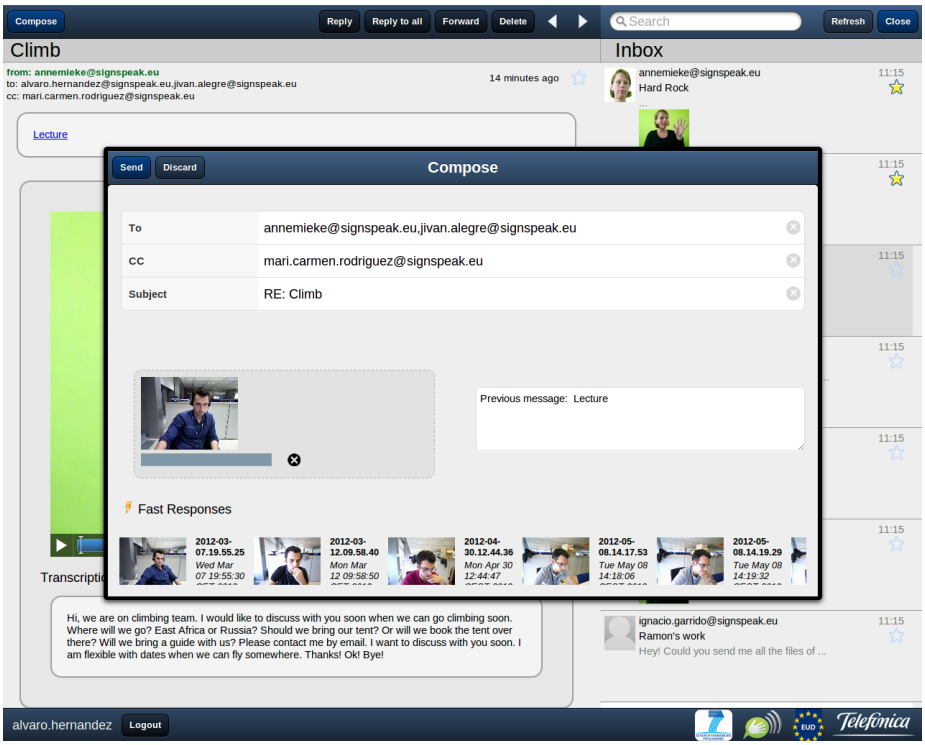

Figura 4.4: Modal de composición de mensajes de VideoSL Mail

adjunto, que no se consideró necesario por la funcionalidad que podría requerir para su funcionamiento.

Los campos *To*, *CC* y *Subject* se implementaron como campos de formulario Sencha, al igual que el campo de texto de la parte central del modal. En estos componentes no hubo ningún problema de estilo relacionado directamente con estos componentes. La parte central izquierda es una zona de arrastre, y la parte inferior es un carrusel del cual se arrastran previsualizaciones a la zona de arrastre.

En la Figura 4.5, podemos observar cómo se usa la funcionalidad *Drag & Drop*. Dicha función está compuesta de dos componentes. El carrusel de abajo, es un carrusel arrastrable horizontalmente en el que se pueden arrastrar las previsualizaciones a la zona de arrastre (*Drop area* de la imagen). Se añadió una barra de progreso ficticia que simularía el avance en la subida del vídeo o el procesamiento del vídeo por parte de Signspeak.

Implementar y hacer funcionar la funcionalidad *Drag & Drop* fue de largo lo más problemático de todo el desarrollo. Para implementar la funcionalidad se encontró un ejemplo de uso en HTML5 Rocks [29]. De esta forma se adaptó el código de dicha transparencia para usarlo en el modal de composición de VideoSL Mail.

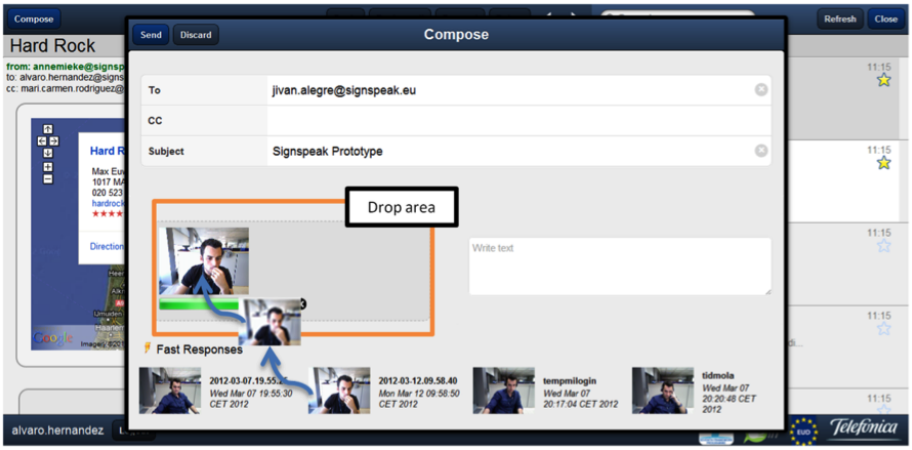

Figura 4.5: Modo de uso para añadir un vídeo a un mensaje en VideoSL Mail

Haciendo uso de Sencha Touch, no existen las típicas barras de desplazamiento para navegar por la aplicación web ya que está pensado para su uso en Smartphones o Tablets, todos los paneles son arrastrables. Uno de los principales problemas de todo el Frontend se encontró al implementar las zonas de Drag & Drop en componentes Sencha. El arrastre de las imágenes de previsualización del vídeo colisionaba con el scroll interno que efectuaba Sencha Touch por su cuenta. Es decir, se arrastraba una imagen, se podía soltar en la *Drop area*, pero a efectos de scroll horizontal en el carrusel se quedaba atascado. Para arreglarlo bastaba con pulsar sobre cualquier parte del modal, pero no era un comportamiento deseable.

Haciendo uso del inspector de Google Chrome, se empezaron a buscar funciones o atributos del carrusel Sencha para intentar arreglar este comportamiento. Así se consiguió que cuando se soltara cualquier imagen sobre la *Drop zone*, al manejar el evento, se le comunicara al panel del carrusel que dejara de hacer scroll. De esta forma se arregló en gran parte el problema que surgió de tener un panel con scroll Sencha Touch y elementos que pudieran ser arrastrados.

Por último, cuando se pulsara sobre cualquiera de las imágenes de previsualización de vídeos, debería reproducirse el vídeo. Al igual que en la bandeja de entrada, la reproducción de vídeo se realizó usando el tag HTML5 *video*.

4.1.3. Pantalla de autenticación

En la Figura 4.6 se puede ver un boceto para pantalla de autenticación del servicio VideoSL Mail. Esta pantalla fue realizada por un compañero con una página HTML y un archivo php que comprobaba si se introducían un usuario y contraseña preestablecidos. No se profundizó en ningún modo esta parte del Frontend.

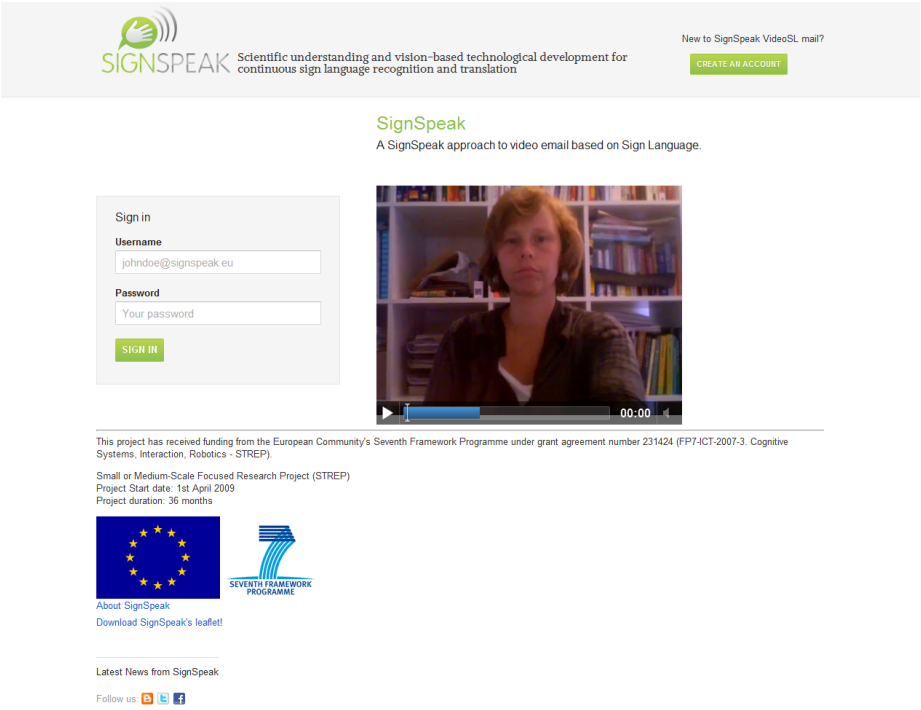

Figura 4.6: Mockup de la pantalla de autenticación en la aplicación

4.2. Primera versión del Backend y adaptación del Frontend

4.2.1. Desarrollo inicial del Backend

Para el desarrollo del Backend ya se tenía una base de otro servidor completo usado en otro proyecto. Este servidor ya tenía configurado Jersey para la interfaz REST y la base de datos eXist para la persistencia de datos.

El desarrollo del Backend y la administración del servidor Tomcat se realizó a través de Eclipse EE. De esta forma se importó al proyecto el paquete *war* correspondiente a la base de datos eXist y el paquete que contenía los servlets del proyecto del que se partía de base. Por otro lado, todo el código del Frontend se copió a la carpeta webapps del servidor Apache Tomcat 7.0; de esta forma, se simplificaban aspectos de seguridad como *Cross-origin*.

Una vez el contenedor eXist había sido desplegado. Se empezó con la configuración inicial de la base de datos. Para tener ciertos datos de ejemplo siempre disponibles, se importó un archivo XML a la base de datos. La administración de la base de datos eXist tenía una interfaz web a través de la cual se podían realizar ciertas operaciones. Una de ellas era la importación de archivos XML dentro de la base de datos. Este archivo definirá la estructura general de la base de datos.

Listing 4.4: Contenido inicial de la base de datos

```
<signspeak_mail>
        <user name="[user]@signspeak.eu" thumbnail="[thumbnail_img
            ].png">
                <mail>
                        <message>
                                <from>[firstsender]@signspeak.eu</
                                    from>
                                <to>
                                        <adressee type="recipient">[
                                            user]@signspeak.eu</
                                            adressee>
                                        <adressee type="cc">[user2]
                                            @signspeak.eu</adressee>
                                </to>
                                <labels>
                                        <label name="Label1" color="#[
                                            color1]"/>
                                </labels>
                                <tags>
                                        <starred/>
                                        <important/>
                                        <unread/>
                                </tags>
                                <subject>
                                        [subject]
                                </subject>
                                <body>[body_text]</body>
                                <videos>
                                        <video_transcription filename=
                                            "[video_url].mp4" thumbnail
                                            ="[video_thumbnail_url].png
                                            ">[video_transcription]</
                                            video_transcription>
                                </videos>
                        </message>
                </mail>
        </user>
</signspeak_mail>
```

En el Listing 4.4, podemos ver el esquema general del contenido de la base de datos. Tendremos tantos elementos *user* como usuarios existan, cada uno tendrá asociado un avatar (atributo *thumbnail*), y cada usuario tendrá unos mensajes asociados. Estos mensajes son únicamente mensajes recibidos. Cada elemento *mail* contiene un sólo elemento *message*, y este a su vez contiene un remitente (*from*), y tantos destinatarios (*adressee*) como se quieran. Estos destinatarios tienen definido su tipo y el contenido del elemento es su identificador. Pueden tener etiquetas (*labels*), y bajo *tags* se encuentran tres posibles propiedades. Si el elemento ha sido marcado como favorito, tendrá un elemento *starred*, e igual si se ha marcado como importante (*important*) o todavía no se ha leído (*unread*).

Por último, el elemento *subject* contiene el asunto, *body* el cuerpo del mensaje y *videos* la información necesaria para mostrar y reproducir el vídeo grabado en lengua de signos.

Una vez llegados a este punto, es hora de implementar los métodos de la arquitectura REST para que el Frontend pueda conectarse y conseguir información del Backend. De esta forma tenemos por un lado código Java que se ejecutará al recibir alguna petición HTTP a los métodos definidos, y cuando sea necesario, se hará una petición a la base de datos XML. Todas las URLs están contenidas en el servlet *signspeakTIDMod* bajo la ruta api y son accesibles a través de la ip y el puerto del servidor Tomcat. Esta ip y puerto está definida en la URLs como BACKEND_URL. De esta forma todas las URLs tendrán el siguiente esquema:

http://BACKEND_URL/signspeakTIDMod/api/method?parameter1=value1

4.2.1.1. Método GET getMessages

Este método será consultado por el Frontend nada más arrancar, le pedirá los mensajes del usuario USER al Backend y este tendrá que devolvérselos. La URL en la que se recibirá la petición será:

$http://BACKEND\_URL/signspeakTIDMod/api/getMessages?username=USER$

En este caso únicamente se hace una consulta XQuery a la base de datos y se devuelve directamente el resultado. La consulta XQuery es la siguiente:

$//user[@name="USER@signspeak.eu"]/mail/message$

De esta forma las dos primeras barras verticales indican que se profundice lo necesario hasta encontrar todos los nodos *user*, dentro de todos ellos, se filtran aquellos que contengan un atributo (el filtro por atributo se designa con la @) *name* que sea igual al usuario de la base de datos. Y dentro de ese usuario, se devuelven todos los elementos *message* que contenga.

4.2.1.2. Método GET search

Este método realizará una búsqueda de la cadena de texto SEARCH dentro de la bandeja de entrada del usuario USER definido. La URL definida para este método es:

$http://BACKEND\_URL/signspeakTIDMod/api/search?username=USER\&search=SEARCH$

De esta forma, tal como en el apartado anterior, se ejecutará una consulta a la base de datos y se devolverá el resultado. Nada más.

$//user[@name="USER@signspeak.eu"]/mail/message[contains(.,'SEARCH')]$

Esto filtra todos los mensajes del usuario USER que contengan la cadena de texto SEARCH.

4.2.1.3. Método GET fastresponses

Este método devolverá una lista de los vídeos subidos por el usuario con el VideoUploader. En esta etapa del desarrollo, los vídeos se han puesto manualmente sobre una carpeta *videos* dentro de la carpeta webapps del servidor Apache Tomcat. De esta forma cada vídeo es accesible a través de:

$http://BACKEND\_URL/videos/USER/VIDEO.mp4$

Cada vídeo tiene una previsualización con el mismo nombre. Así, cuando este método se invoque, se listará el directorio de vídeos del usuario, filtrará los archivos de imagen y compondrá un JSON que sigue el formato del Listing 3.4. Como se puede ver, en *fastresponses* se devuelve únicamente la URL de la imagen, para conseguir la url del vídeo únicamente hay que sustituir la extensión de la imagen por mp4.

4.2.1.4. Método POST sendMail

En este caso desde el Frontend se enviará a través de una petición POST un elemento Message en XML que contenga toda la información del mensaje a enviar, lista para ser insertada en la base de datos.

De esta forma, se extraerán del mensaje todos los destinatarios, y se añadirá el mensaje a la bandeja de entrada de cada destinatario. Para hacer esto se ejecuta esta consulta por cada destinatario USER:

$update\ insert\ <mail>XML\_DATA</mail>\ into\ //user[@name="USER"]$

Una vez completados estos pasos se ejecuta la transcripción del vídeo. Como en ningún momento se ha dispuesto del software de la plataforma Signspeak, siempre se ha hecho todo de forma simulada. Por ahora como no disponemos de Cloud Infraestructure, se le asigna una transcripción estática. Para añadir

dicha transcripción a las referencias a los vídeos de la base de datos, se ejecuta la siguiente consulta:

$update\ value\ //video\_transcription[@filename = "URL"]\ with\ TRANSCRIPTION$

Esta consulta busca todos los elementos *video_transcription* y modifica el contenido del nodo sustituyéndolo con TRANSCRIPTION.

4.2.2. Adaptación del Frontend para su conexión con el Backend

Una vez que tenemos el Backend preparado para recibir peticiones del Frontend ya podemos adaptar este. Para todas las peticiones HTTP, se usó AJAX nativo (XMLHttpRequest), de esta forma se intentó evitar el uso de las funciones AJAX de Sencha Touch o jQuery. Por otro lado, debido a diversas confusiones, en un inicio se usaron peticiones HTTP bloqueantes. De esta forma se simplificó mucho la implementación de las peticiones HTTP, pero por otro lado, la ejecución del JavaScript se detenía hasta conseguir una respuesta del Backend. Estaba planificado cambiar este comportamiento para que fuera asíncrono (y verdaderamente AJAX), pero por falta de tiempo no se pudo finalmente realizar este cambio.

4.2.2.1. Método getMessages

Una vez hecha la petición al Backend de los mensajes del usuario conectado, se recibía un XML contenedor de todos los mensajes. En este punto, se requería procesar este XML para construir las dos listas que se manejaban desde la interfaz: *dataMsg* y *msgsList*. Para procesar el XML se hizo uso del estándar *DOMParser*. Una vez que el árbol XML estaba construido a partir de la respuesta del Backend, el acceso a elementos era muy similar al acceso de elementos HTML.

Una vez creado el árbol XML a partir de la respuesta gran parte de la construcción de los nuevos *dataMsg* y *msgsList* fue trivial.

4.2.2.2. Método search

Cuando se escribía algo en la barra de búsqueda del Frontend y se recibía un evento *action* (usualmente pulsar retorno de carro) se lanzaba la búsqueda. Al procesar la respuesta, primero se comprobaba si se había recibido una respuesta vacía, en cuyo caso se mostraba un mensaje para indicar que no se han encontrado resultados. Si la respuesta no estaba vacía, se mostraban en la lista de mensajes los mensajes encontrados.

Para volver a la bandeja de entrada, era necesario borrar el contenido de la barra de búsqueda y pulsar retorno de carro.

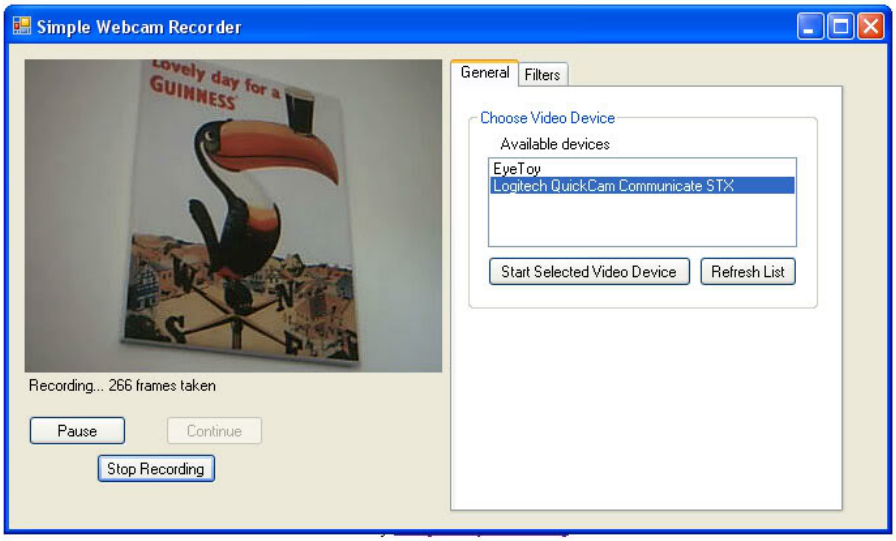

Figura 4.7: Interfaz de simple-webcam-recorder

4.2.2.3. Método fastresponses

El Frontend hacía una petición HTTP con este método siempre que se cargaba el modal de composición de mensajes. En este caso no se requería de ningún procesamiento intermedio, ya que la respuesta que se recibía del Backend es JSON. Haciendo uso de la función *eval*, se consigue crear un objeto JavaScript a partir del texto que se le pase como argumento. De esta forma haciendo uso de eval a la respuesta recibida, ya teníamos lista nuestra variable *fastresponses*.

4.2.2.4. Método sendMail

Es necesario componer un XML con los datos del mensaje compuesto para que sea enviado al Backend. Aunque semánticamente se tratara de un XML, el mensaje a enviar se construyó manualmente como una cadena de texto.

En el caso de que no hubiese escrito nada en el campo *to*, se avisa al usuario de que debe insertar algún destinatario. En cualquier otro caso se proseguía con el envío y se cerraba el modal de composición.

4.3. Desarrollo independiente del Video Uploader

El desarrollo del Video Uploader dio muchos menos problemas de los previstos. Lo primero que se realizó fue una adaptación del interfaz gráfico de simple-webcam-recorder. El interfaz del que se partía se puede ver en la Figura 4.7.

Este interfaz se adaptó hasta conseguir el diseño final del Video Uploader. Este diseño final se puede ver en la Figura 4.8. Como se puede observar, se

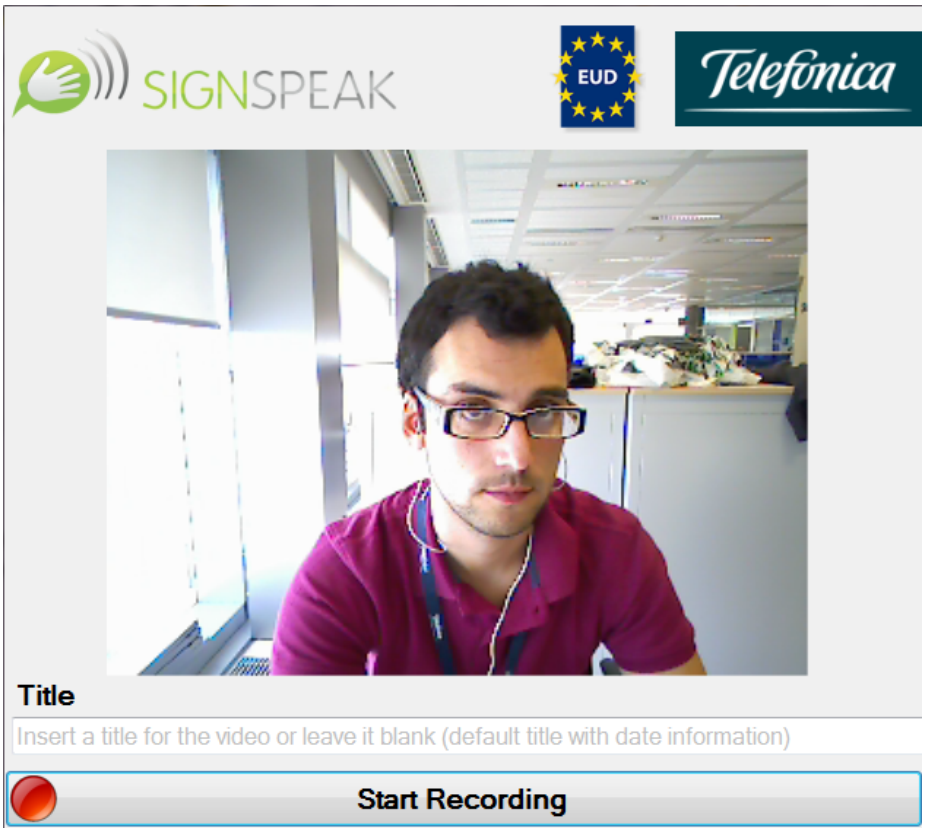

Figura 4.8: Interfaz de la pantalla principal del Video Uploader

eliminaron todos los controles que no fueran iniciar grabación y parar. De esta forma no permite filtros, la selección de cámara es automáticamente la primera cámara disponible y se amplió la superficie de previsualización de cámara hasta ocupar casi toda la ventana. Se añadió un campo de texto en caso de que el usuario quisiera añadir un título al vídeo. Por último se añadieron los logos del las compañías implicadas en el proyecto.

Además, se añadió una falsa pantalla de autenticación, la cual se puede ver en la Figura 4.9. En ningún caso se comprueba el usuario o la contraseña contra el Backend, simplemente se guarda el usuario insertado para un futuro uso y cambia a la pantalla principal para comenzar con la grabación de vídeo.

Una vez terminada la parte gráfica se inició el trabajo lógico. Para empezar, era necesario generar una previsualización del vídeo grabado. Por eso, se modificó el código de la grabación de vídeo para que, después del primer segundo, uno de los *frames* capturados fuera a parar a un fichero BMP en el mismo directorio donde se grabó el vídeo y con el mismo nombre.

Cuando se hiciera click en *Stop Recording*, se lanzaría en segundo plano un terminal (invisible para el usuario) que ejecutaría *FFmpeg* para la conversión del

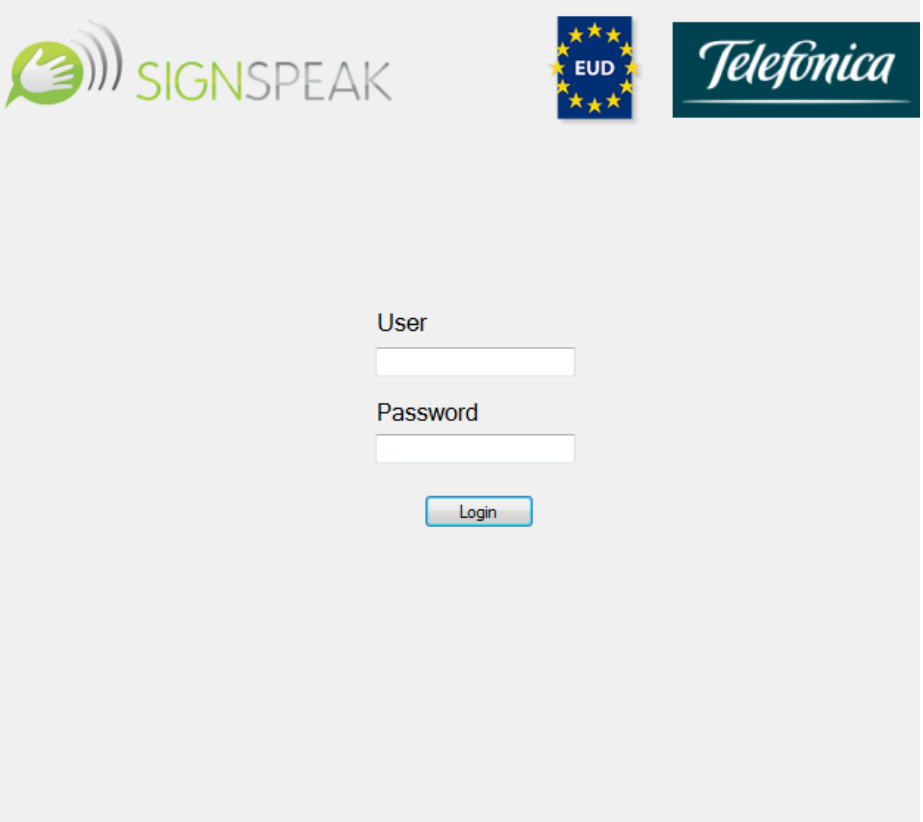

Figura 4.9: Pantalla de autenticación de Video Uploader

vídeo a un formato compatible con la web. Para ello, se establecía como directorio de trabajo la carpeta donde se han ido guardando los vídeos. La ruta del binario *FFMpeg* (FFMPEG_PATH) se tendría guardado como parámetro. De esta forma el comando a ejecutar es:

Listing 4.5: FFmpeg para convertir vídeo

```
FFMPEG_PATH\ffmpeg.exe -i VIDEO.avi-acodec libfaac -vcodec libx264
    -r 10 VIDEO.mp4
```

Una vez terminado, borra el vídeo AVI original y ya estaría preparado para su conexión con el Backend.

4.4. Actualización del Backend y conexión de Video Uploader con éste

En esta sección cabe destacar que surgieron varios problemas graves que hicieron modificar la arquitectura planeada inicialmente. El más grave fue la imposibilidad de hacer reconocer el mensaje compuesto en el Video Uploader por el

Backend. Es decir, en un inicio se compuso una trama HTTP multipart estándar, pero o bien porqué estaba mal compuesta en el Video Uploader, o bien porqué se procesaba inadecuadamente en el Backend, nunca llegaba a reconocer bien la trama enviada. De esta forma se compuso una trama alternativa en la que directamente se escribían bytes como texto en la trama HTTP.

Cabe recordar que el esquema de la trama HTTP que envía el Video Uploader se puede ver en la Tabla 3.1.

4.4.1. Envío de datos por parte del Video Uploader

Para las peticiones HTTP en C# se usaron objetos HttpWebRequest. Primero se creó el objeto con la URL completa del método upload del Backend. Después se añadieron el nombre de usuario y el nombre del vídeo a la cabecera HTTP.

Una vez compuesta la cabecera, se escribe en el *Stream* de la petición HTTP el separador *–video–*. Después, los bytes del vídeo grabado y listo para la web (aquel convertido con FFmpeg). De nuevo se vuelve a escribir *–video–* y después los bytes de la captura realizada.

Una vez se han realizado todos estos pasos, el vídeo se ha enviado al Backend para que se guarde y se muestre en el Frontend.

4.4.2. Método upload en el Backend

Fue complicado procesar la trama HTTP enviada por el Video Uploader, ya que mezclaba en un mismo mensaje texto plano con información relevante, un separador en texto plano, bytes del vídeo, otro separador en texto plano y finalmente los bytes de la imagen.

Para separar la trama entrante en partes usando el separador *–video–* se decidió tratar toda la trama como bytes. De esta forma la trama entera, se dividió usando como separador los bytes correspondientes al String *–video–* en ISO-8859-1. La detección de la codificación adecuada se efectuó primero con UTF-8, pero se vio que era incorrecta. De esta forma, teníamos una lista en la que el primer elemento son los bytes asociados a la cabecera HTTP, el segundo elemento son los bytes del vídeo y el último los bytes de la imagen.

Se construyó un String a partir del primer elemento y de ahí se extrajeron el usuario y el nombre del vídeo. Como cada usuario dispone de una carpeta en el Backend donde se almacenan todos sus vídeos, primero se comprueba si esa carpeta existe. Si no existe (el usuario todavía no había subido ningún vídeo) se crea. Una vez hecho esto, se escriben los bytes del vídeo en un archivo *mp4* y los bytes de la imagen en un archivo *bmp*. El título de ambos ficheros será el nombre del vídeo recibido.

De esta forma ya tenemos los vídeos y las previsualizaciones guardados para que cuando el Frontend haga una petición a *fastresponses*, el Backend responda con los nuevos vídeos recién subidos.

4.5. Desarrollo de Cloud Infraestructure

Una vez tenemos el Frontend y el Video Uploader con funcionalidad completa y funcionando, falta simular que varias máquinas puedan transcribir vídeos de VideoSL Mail usando Signspeak. Cabe recordar que la Cloud Infraestructure está compuesta de un sólo Scheduler y uno o varios Slaves que ejecutan el trabajo.

Ya que disponer de 3 máquinas físicas (1 Scheduler y 2 Slaves) para simular la Cloud Infraestructure era muy complicado y difícil de manejar, se optó por virtualizar la infraestructura. De esta forma se usaron máquinas virtuales VirtualBox con Ubuntu y configuradas en red en modo *bridge*. De esta forma, a efectos de red, cada máquina virtual era igual que una máquina física. Así, tanto la máquina huésped como las máquinas virtuales tenían una IP de la misma red.

Para que todo funcione según lo previsto, el Scheduler siempre debe ser el primero en estar activo. Una vez arrancado, recibirá peticiones de nuevos Slaves para mostrarse como máquinas disponibles para procesar vídeos. De esta forma, cuando un Slave arranque (o decida estar disponible para recibir peticiones de transcripción), enviará una petición HTTP al Scheduler al método *newSlave*. Así, cualquier Slave se podrá conectar en cualquier momento, y el Scheduler lo añadirá a su lista de Slaves disponibles.

Paralelamente a todo este proceso, el Backend puede hacer una petición de transcripción invocando al método *transcript* del Scheduler. Hecho esto, el Scheduler pedirá a cada Slave activo su uso de CPU haciendo una petición al método *cpu* de cada uno. Al que menor uso de CPU registre, se le asignará la tarea de la transcripción invocando a su método *transcript*. Esta petición quedará bloqueada hasta que la transcripción por parte del Slave esté completa. Una vez terminada la transcripción y el Scheduler obtenga la transcripción, este último hará una consulta al método *transOK* del Backend.

4.5.1. Scheduler

4.5.1.1. Método newSlave

Para obtener la dirección IP de la máquina que realizaba la petición HTTP se tuvieron bastantes problemas. Jersey permitía una extracción rápida de parámetros introducidos por CGI, pero por otro lado, extraer elementos de más bajo nivel fue bastante más complicado. Al final se pudo extraer la petición HTTP

como una variable Java y a partir de ahí sacar la IP de origen. Una vez sacada la IP y guardarla en una lista en memoria fue trivial.

4.5.1.2. Método transcript

Como el procesado de una petición a este método es bloqueante, para cada petición se creará un hilo nuevo y se devolverá siempre "Transmission queued".

Para realizar peticiones HTTP desde Java se han usado las clases URL y URLConnection. De esta forma, una vez dentro del hilo recién creado, se realizan consultas de CPU a cada Slave invocando a su método *cpu*. Al Slave con menor uso de CPU se le hace una petición al método *transcript* de ese Slave. El hilo del Slave quedará bloqueado hasta que no se reciba alguna respuesta por parte del Slave.

Cuando se reciba respuesta se le hará una petición al Backend en la que irá contenida la URL del vídeo como parámetro y el contenido de la transcripción recibida como cuerpo del mensaje HTTP POST.

4.5.2. Slave

Además de los métodos para las peticiones HTTP que pueda recibir, se ha de dar de alta como Slave al arrancar. Para se creó un método interno que realizaba este alta cuando el servidor iniciaba su ejecución.

4.5.2.1. Método cpu

Devuelve el resultado de ejecutar:

$ManagementFactory.getOperatingSystemMXBean().getSystemLoadAverage();$

Es necesario matizar que este comando sólo funciona en sistemas operativos basados en Unix.

4.5.2.2. Método transcript

Como ya se dijo anteriormente, en ningún momento del desarrollo se dispuso del software para ejecutar realmente Signspeak. Toda la plataforma es un prototipo de un caso de uso de Signspeak. Por esto, para simular el procesamiento de vídeos, se duerme el hilo durante 5 segundos y después de devuelve siempre la misma respuesta:

> In a place of La Mancha of which name I don't want to remember...

4.6. Actualización del Backend para la conexión con la Cloud Infraestructure

Para adaptar el Backend y que usara la simulación de Cloud Infraestructure se tuvo que modificar el método *transcript* para que pidiera la transcripción al Scheduler. Cuando estuviera disponible se recibiría una petición al método *transOK*.

El método *transOK* recibe la URL del vídeo como parámetro y la transcripción recibida del vídeo dentro del cuerpo de la trama HTTP. Luego actualiza los elementos *video_transcription* de la misma forma que antes se hacía aplicando una transcripción estática definida en el Backend (ver Subsubsección 4.2.1.4).

4.6.1. Despliegue de Cloud Infraestructure en una infraestructura real

Una vez desarrollado todo el software, se podría plantear el despliegue de la infraestructura realizada en uno de los servicios vistos en la . Aunque dentro del proyecto no se realizó, no está de más ver las posibilidades de las que dispondríamos.

Una primera solución sería un despliegue en máquinas propias. Sería la aproximación realizada en realidad. Simplemente se despliegan los servidores en máquinas virtuales de una máquina física o sobre servidores de la red de Telefónica I+D. Como ya se ha dicho, finalmente se desplegó en máquinas virtuales sobre una máquina física (HP EliteBook).

En el caso de desplegar la Cloud Infarestructure en un servicio Cloud Computing, sería necesario usar un servicio *IaaS*. Necesitamos acceso total a las máquinas para poder desplegar los servidores. Parte de las soluciones *PaaS*, requieren desarrollar desde un inicio de acuerdo a su framework.

Mención aparte requiere RedHat Openshift, ya que aunque sea una solución *PaaS*, nos permite desplegar sobre Apache Tomcat, por lo que se debería poder desplegar la infraestructura sobre su servicio.

Parte III

Discusión y conclusiones

Capítulo 5

Análisis de resultados y evaluación

Para el análisis de resultados se recurrió a la EUD (*European Union of the Deaf*). La EUD es una organización sin ánimo de lucro que funciona en los 27 países miembros de la Unión Europea más Islandia, Suiza y Finlandia. Su función principal es conseguir igualdad en la vida pública y privada para la comunidad sorda.

Es necesario matizar, que gran parte del trabajo desarrollado en este capítulo ha sido realizado por Álvaro Hernández-Trapote. Mi trabajo se centró más en realizar un guión de uso de la herramienta para enviarlo a los usuarios.

Antes de analizar la evaluación de la experiencia de usuario, cabe que se recibieron algunas ideas acerca de VideoSL Mail antes de realizar la evaluación. En una ocasión nos dijeron que la mayoría de usuarios sordos preferían tener la lista de mensajes a la derecha de la pantalla. De esta forma en la parte izquierda queda la vista detallada y en la derecha la lista.

5.1. Dispositivos

Se envió un dispositivo Acer Iconia W500 (ver Subsección 3.4.2) a la sede de la EUD en los Países Bajos. El objetivo de las pruebas era conseguir retorno de experiencia para las partes del cliente. De esta forma, se usó una versión sin Cloud Infraestructure, ya que no comporta ninguna mejora real de cara al usuario final. Lo que si se usó fue un Backend modificado en la tablet. Así los cambios en el Frontend no surtirían efecto para posteriores ejecuciones, y todos verían los mismos mensajes. De esta forma se independizaba de el portátil que usaba

para desarrollo y únicamente era necesaria la tablet para la ejecución de todo el software de la parte del Cliente.

5.2. Objetivos

Con esta evaluación de experiencia se pretendía:

- Encontrar problemas de usabilidad. No sólo del software desarrollado sino posiblemente de los dispositivos en sí
- Conseguir información subjetiva acerca de la apariencia de la aplicación, su diseño o su rendimiento
- Determinar si el servicio logra atraer la atención del usuario, si el cliente pagaría por un servicio así
- Sentimiento general del servicio

5.3. Pasos para usar VideoSL Mail

En el Apéndice A se puede ver el formulario que se le facilitó a la EUD para correcta evaluación del servicio por cada usuario. Por otro lado, se añadió un consentimiento para que no hubiera problemas de privacidad acerca de los datos conseguidos a través de esta evaluación de experiencia. Este consentimiento se puede ver en el Apéndice B.

Los pasos a realizar para la evaluación son:

- Autenticarse: El objetivo de esta acción, es comprobar cómo se adapta el usuario al teclado virtual de la tablet Iconia
- Buscar: Buscar un mensaje en la barra de búsqueda. Una vez lo encuentren, borrarlo. Con esto interactuarán con los paneles arrastrables y los mensajes
- Crear un mensaje: Deben probar la funcionalidad Drag & Drop a través del carrusel de previsualizaciones
- Salir de la aplicación

5.4. Base de usuarios

La aplicación fue probada por 5 personas (3 mujeres y 2 hombres). El rango de edad se sitúa entre los 23 y 38 años y su origen en Canadá, Dinamarca, Países Bajos (2) y España.

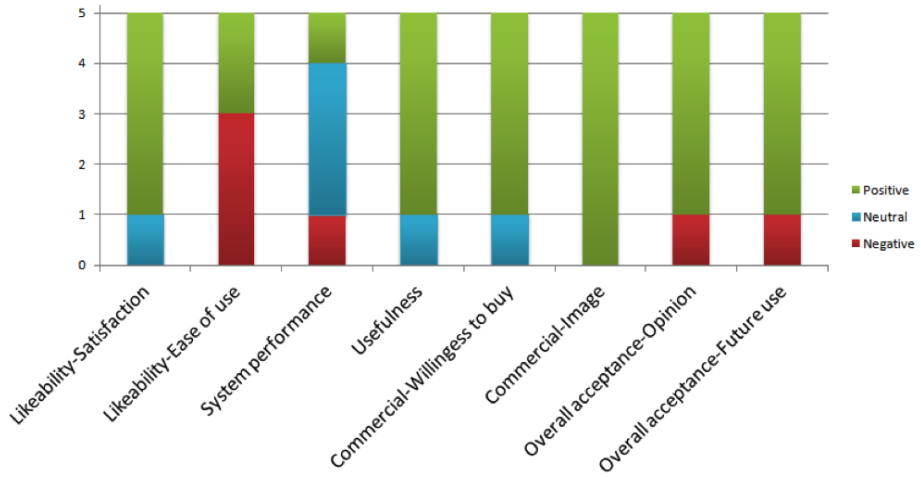

Figura 5.1: Gráfica de usabilidad

Parece una cifra pequeña, pero 5 participantes en una evaluación de usabilidad son suficientes para encontrar el 85 % de problemas.

5.5. Resultados

En los resultados podemos destacar los siguientes puntos:

- Simpatía: En general a la mayoría de los usuarios les gusta el servicio de VideoSL Mail. Pero tienen problemas para usar la interfaz, ya que tres de ellos han valorado negativamente la facilidad de uso
- Rendimiento del sistema: Resultado neutral, tres de los usuarios votaron positivamente y dos de ellos dieron un voto negativo
- Usabilidad: Cuatro de los usuarios lo encontraron usable
- Visión comercial: Cuatro usuarios decían que pagarían por tener un servicio como VideoSL Mail.
- Opinión general: El servicio ha tenido una aceptación muy positiva ya que 4 de los usuarios están contentos con el servicio.

En la Figura 5.1 podemos observar estos resultados de forma gráfica.

Entre las ventajas encontradas por los usuarios están:

- Interfaz clara y fácil de usar
- Amigable para sordos

- Adjuntar vídeos con Drag & Drop es fácil y cómodo
- Una vez acostumbrado, todo se puede realizar de forma rápida
- La visualización de vídeos a partir de las capturas es muy útil

Entre los puntos negativos:

- Problemas con el teclado, pequeño e incómodo
- Funcionalidad limitada
- Sensibilidad de la pantalla muy reducida
- Las capturas de vídeos deberían ser mas grandes
- Carencia de una lista de contactos
- Interfaz del Video Uploader bastante pobre
- Sólo se puede añadir un vídeo por mensaje
- Debería ser posible grabar vídeos desde la interfaz web

5.6. Conclusiones de la evaluación

Como podemos ver, los usuarios disfrutaron VideoSL Mail, pero le encontraron varios problemas de usabilidad.

En general, VideoSL Mail fue considerado como una idea muy buena para la comunidad sorda. Incluso consideraban que, aunque era un valor añadido, no era indispensable la plataforma Signspeak para la aplicación. Consideraban que un cliente de videomensajería adaptado a las necesidades de los sordos ya cumplía las expectativas.

Capítulo 6

Conclusiones y líneas futuras

6.1. Presupuesto

Este proyecto ha sido presupuestado de forma real. Pero debido a mi condición de becario, no tengo acceso a esos presupuestos. Es por ello por lo que he decidido no exponer ningún tipo de presupuesto, ya que además de ser un presupuesto inventado para un proyecto real, posiblemente se aleje bastante de la realidad.

6.2. Conclusiones

6.2.1. Éxito en los objetivos

La meta del proyecto era:

> Crear un software de videomensajería en la nube que use la plataforma Signspeak.

En general, podemos decir que se ha creado el software requerido, no es seguro ni está preparado para desplegarse en una infraestructura real como producto, pero se ha llegado a un buen resultado para tratarse de un prototipo.

En cuanto a los objetivos, se han cumplido en mayor o menor manera todos ellos. Primero se ha diseñado una arquitectura. En ella se ha considerado ya desde el inicio que el Video Uploader era un componente fundamental. Se han separado Frontend y Backend en dos componentes independientes. Por último se ha decidido desarrollar una Cloud Infraestructure que forma parte de la solución global. En este apartado considero que no ha habido un éxito absoluto, ya que considero que el Video Uploader habría sido un componente completamente prescindible si

se hubiera usado el API HTML5 MediaCapture. Habría sido una solución mucho más elegante y fácil de usar que la realizada.

El segundo objetivo era definir las tecnologías a usar, y se ha optado por reutilizar software ya desarrollado para el caso Backend. Para el resto de componentes se han usado las tecnologías que en principio menos tiempo requerían para tener el software desarrollado. Estas tecnologías han cumplido con lo esperado.

El tercero es definir las interacciones entre componentes. En este caso aunque las comunicaciones no sean seguras, creo que se ha desarrollado un buen trabajo. Se han especificado desde el inicio las comunicaciones logrando una buena estructura a la hora de desarrollar y documentar.

En el desarrollo del software se han cumplido muy satisfactoriamente las expectativas. Se ha desarrollado un software bastante robusto para lo requerido y con una funcionalidad bastante extensa. El interfaz web es muy amigable y el Video Uploader aunque no lo sea tanto, es muy fácil de usar.

El último objetivo es la evaluación por parte de la EUD. En este caso creo que se ha dado una buena cobertura a los resultados de los formularios.

6.2.2. Conocimientos de la carrera aplicados

Los conocimientos adquiridos en la carrera y que he aplicado en el proyecto son más generales que concretos. Por ejemplo, en la carrera no había dado nada de JavaScript en el inicio del desarrollo, lo que hizo que tuviera que aprenderlo desde cero. Cuando se enseñó algo de AJAX en Servicios y Aplicaciones Telemáticas, ya estaba gran parte del desarrollo realizado.

Para el desarrollo de los servidores sí que me sirvió de mucho la asignatura Sistemas Telemáticos II. En ella aprendimos a desarrollar un servidor en Java usando Servlets, lo cual es muy parecido al desarrollo de Backend y Cloud Infraestructure.

En cuanto al desarrollo del Video Uploader, no había desarrollado nada en C#, pero debido a la similitud con Java no fue difícil.

Con el desarrollo de todo el servicio VideoSL Mail, la asignatura de Ingeniería del Software adquirió otro sentido. Me di cuenta de que para simplificar las cosas en un futuro, es necesario planificar todo lo necesario para que no hayan sorpresas a la hora de desarrollar.

6.3. Líneas futuras

En el caso de que se pudiera seguir profundizando en el proyecto, algunas líneas sobre las que se podría realizar trabajo son:

- Desarrollo para otras plataformas: Considero esencial hacer que VideoSL Mail funcione desde un terminal Android o iPhone. Con el API HTML5 MediaCapture debería ser casi trivial
- Realizar un nuevo Video Uploader para sistemas operativos de escritorio: Rediseñar el VideoUploader para que codifique directamente vídeo en AVC sin necesidad de FFMpeg. Por otro lado considero necesario que funcione en otros sistemas operativos como Mac OS X o GNU/Linux
- Añadir seguridad: De esta forma se podrían hacer consultas de forma segura, con autenticación y registro de usuarios.

Parte IV

Apéndices

Apéndice A

Apéndice 1

“VideoSL mail” Evaluation Protocol

Participant Name:_____

Date:_____

☐ **Hearing** ☐ **Deaf**

☐ **Welcome**

☐ **Instructions**

☐ **Consent form**

☐ **Adjust videocamera**

☐ **Start video recording**

Test block 1: Interaction with the service

Task 1: Sign-in

- Please sign-in to the service. (User: alvaro.hernandez; Pwd: password)

Task 2: Search

- Please search in your inbox a video message containing the word “lecture”
- Check the encountered message
- Delete the message

Task 3: Compose

- Please compose a new message to Javier (test@signspeak.com) using the “fast responses” feature
- Send the message

Task 4: Sign-out

- Please sign out from the service

☐ Pause video recording

My experience with the user:

☐ The user showed enthusiasm

☐ The user had many problems using the system

☐ I helped the user during the interaction

Others:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Test block 2: Questionnaire

Previous Experience

1. Please, mark your previous experience regarding to

   a. eMail services: 1 (Null) ☐ 2 ☐ 3 ☐ 4 ☐ 5 (Expert)☐

   b. Tablet PC devices: 1 (Null) ☐ 2 ☐ 3 ☐ 4 ☐ 5 (Expert)☐

Likeability

2. I like this service Strongly disagree ○ ○ ○ ○ ○ Strongly agree

3. It is easy to use Strongly disagree ○ ○ ○ ○ ○ Strongly agree

System performance

4. The system works well Strongly disagree ○ ○ ○ ○ ○ Strongly agree

Usefulness

5. This service is useful Strongly disagree ○ ○ ○ ○ ○ Strongly agree

Willingness to buy

6. I would pay for this service Strongly disagree ○ ○ ○ ○ ○ Strongly agree

7. I like companies which have this kind of services within their products

   Strongly disagree ○ ○ ○ ○ ○ Strongly agree

Overall acceptance

8. My overall opinion about the system is Strongly negative ○ ○ ○ ○ ○ Strongly positive

9. I would use this service Strongly disagree ○ ○ ○ ○ ○ Strongly agree

10. Please, could you indicate 3 advantages and 3 disadvantages of this system?

11. Is this system good for Deaf community to make a bridge with hearing people? If so why? If not, why?

12. Would you like to suggest any change? (e.g. including a video-recording feature for composing new messages, etc.)

Additional comments:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Apéndice B

Apéndice 2

ANNEX B

Consent form for video recording for research purposes

SIGNSPEAK: Scientific understanding and vision-based technological development for continuous sign language recognition and translation"

Project funded from the European Community's Seventh Framework Programme under grant agreement nº 231424 (FP7-ICT-2007-3. Cognitive Systems, Interaction, Robotics - STREP).

Partners involved in this study: Telefónica I+D, EUD (European Union of the Deaf)

Place of video recording: _____

Date of video recording: _____

Participants's name: _____

This study involves the audio and video recording of your interview with the researcher. Neither your name nor any other identifying information will be associated with the video recording or the transcript. Only the research team will be able to analyse to the recordings. Transcripts and data extracted from your interview may be reproduced in whole or in part for use in presentations or written products that result from this study. It is understood that this material will be used in a legitimate manner, within the context of the SignSpeak project and is not intended to cause any harm or undue embarrassment to the parties involved.

By signing this form, I am allowing the researcher to audio or video tape me as part of this research. I also understand that this consent for recording is effective until the end of the SignSpeak project. On or before that date, the tapes will be destroyed.

Participant's Signature: _____

Bibliografía

[1] Signspeak. Proyecto signspeak. Accedido el 2 de Marzo de 2013. http://www.signspeak.eu

[2] EUD. European union of the deaf (eud). Accedido el 25 de Marzo de 2013. http://www.eud.eu

[3] European Comission. Seventh framework programme. Accedido el 2 de Marzo de 2013. http://cordis.europa.eu/fp7/home_en.html

[4] Europa Press. 90Accedido el 16 de Marzo de 2013. http://sid.usal.es/noticias/discapacidad/19321/1-1/el-90-de-sordos-son-analfabetos-funcionales-en-catalunya.aspx

[5] A2Stream Inc. mailvu. Accedido el 25 de Marzo de 2013. http://mailvu.com

[6] Eyejot Inc. eyejot. Accedido el 25 de Marzo de 2013. http://www.eyejot.com

[7] WhatsApp Inc. Whatsapp. Accedido el 25 de Marzo de 2013. http://www.whatsapp.com/?l=es

[8] Spotbros Technologies S.L. Spotbros. Accedido el 25 de Marzo de 2013. http://www.spotbros.com/wordpress/

[9] Naver Inc. Line. Accedido el 25 de Marzo de 2013. http://line.naver.jp/en/

[10] Microsoft. Skype. Accedido el 26 de Marzo de 2013. http://www.skype.com/en

[11] Google.com. Google+ hangout. Accedido el 26 de Marzo de 2013. http://tools.google.com/dlpage/hdngoutplugin

[12] Google Inc. Google Apps for Business. Accedido el 26 de Marzo de 2013. http://www.google.com/intl/es/enterprise/apps/business/

[13] Microsoft. Office 365. Accedido el 26 de Marzo de 2013. http://office.microsoft.com/es-es/?fromAR=1

[14] Google.com. Google App Engine. Accedido el 26 de Marzo de 2013. https://developers.google.com/appengine/

[15] Red Hat. Openshift by red hat. Accedido el 26 de Marzo de 2013. http://www.openshift.com

[16] Amazon.com. Amazon Elastic Compute Cloud. Accedido el 26 de Marzo de 2013. http://aws.amazon.com/es/ec2/

[17] Microsoft. Windows Azure. Accedido el 26 de Marzo de 2013. http://www.windowsazure.com/es-es/

[18] idan.itzhaky@gmail.com. simple-webcam-recorder. Accedido el 9 de Abril de 2013. http://code.google.com/p/simple-webcam-recorder/

[19] AForge.NET. AForge.NET. Accedido el 9 de Abril de 2013. http://www.aforgenet.com

[20] FFmpeg. FFmpeg. Accedido el 9 de Abril de 2013. http://www.ffmpeg.org

[21] About.com. HTML5 Video Format. Accedido el 9 de Abril de 2013. http://webdesign.about.com/od/video/a/html5-video-formats.htm

[22] W3C. HTML Media Capture. Accedido el 18 de Abril de 2013. http://dev.w3.org/2009/dap/camera/

[23] Eric Bidelman, HTML5 Rocks. Capturing Audio & Video in HTML5. Accedido el 18 de Abril de 2013. http://www.html5rocks.com/en/tutorials/getusermedia/intro/#toc-round1

[24] Sencha Inc. Sencha Touch Build Mobile Web Apps with HTML5. Accedido el 18 de Abril de 2013. http://www.sencha.com/products/touch

[25] jQuery. jQuery Mobile. Accedido el 18 de Abril de 2013. http://jquerymobile.com

[26] Apache Software Foundation. Apache Tomcat. Accedido el 18 de Abril de 2013. http://tomcat.apache.org

[27] GlassFish Java. Jersey. Accedido el 18 de Abril de 2013. http://jersey.java.net

[28] eXist Solutions. eXist-db. Accedido el 18 de Abril de 2013. http://exist-db.org/exist/apps/homepage/index.html

[29] Marcin Wichary. HTML5 Presentation, Drag and Drop. Accedido el 26 de Abril de 2013. http://slides.html5rocks.com/#drag-and-drop