Proyecto fin de carrera. Escuela de Ingeniería Informática.
Universidad de Las Palmas de Gran Canaria.

Kairos: Gestión remota de pacientes clínicos

Yeray Santana Benítez
Las Palmas de Gran Canaria

Proyecto fin de carrera de la Escuela de Ingeniería Informática presentado por el alumno:

YERAY SANTANA BENÍTEZ

Título del Proyecto: Kairos, gestión remota de pacientes clínicos
Tutor: Abraham Rodríguez Rodríguez

Dedicatoria

A mis padres, mi hermana y mis abuelos

Agradecimientos

A mi tutor Abraham Rodríguez Rodríguez, por hacer muchas veces el papel de mi conciencia y no dejar que me desviara del camino correcto, con todo el mérito que eso conlleva.

A Daniel González Santana, por ser el cerebro del que surgio la idea y poner su conocimiento a mi disposición.

A mis padres y a mi hermana, por la educación, el cariño y los valores que me han enseñado; este proyecto tiene bastante de ellos.

A mis amigos, por brindarme tantos momentos de alegria entre análisis, diseños y cabezazos contra la pared.

A mis compañeros de universidad, que convirtieron todos esos años en una de las mejores experiencias de mi vida, regalándome un montón de recuerdos que guardo con nostalgia y alegría.

A tantos y tantos compañeros de trabajo con los que he tenido el placer de trabajar y que me han ayudado a convertir en el profesional que soy hoy en día.

Índice

1. Introducción

Con un tiempo medio de espera de 4 horas en las urgencias de los hospitales españoles[1] es lógico pensar que se hace necesaria la incoporación de nuevas herramientas y sistemas tecnológicos que faciliten la atención a los pacientes y que permitan una mejora en la atención sanitaria que reciben.

Ha sido en los Estados Unidos, mercado pionero en el uso de asistencia telemática en el ámbito sanitario, donde se ha producido un auge en el desarrollo de plataformas orientadas a la asistencia online de pacientes. Esto se debe en gran parte, a las elevadas inversiones del sector privado en el terreno de la telemedicina, dándonos una idea, de la importancia que ha ido ganando dicho campo en el futuro de nuestro sector sanitario.

Por lo expuesto anteriormente surge la motivación de la realización de este proyecto. Nuestro objetivo es la creación de un portal destinado a la gestión de citas y consultas médicas mediante asistencia telemática. Un entorno en el que el paciente pueda solicitar citas y recibir atención remota por videoconferencia, además de tener acceso a toda la información generada. Un entorno en el que el facultativo pueda gestionar su calendario de citas, atender consultas online y acceder a la información de los pacientes. Un entorno en el que las entidades del ámbito sanitario puedan ofrecer un sistema de atención con un bajo coste para su negocio y un gran impacto para sus clientes.

Todo lo anterior no sería posible sin un sistema tecnológico que diera soporte a las funciones de comunicación entre dos clientes situados en distintas localizaciones. Es aquí donde surge una tecnología que nos permitirá implementar un sistema de interacción por video y chat de garantías; una tecnología que aporta un conjunto de ventajas sobre las existentes hasta el momento en el mercado: hablamos de WebRTC o Web Real-Time Communication.

Empezaremos la redacción del presente documento estableciendo los objetivos del proyecto, para después presentar algunas de las herramientas más significativas dedicadas a la asistencia telemática. Elaboraremos un plan de negocios que haga que nuestro proyecto sea atractivo para los clientes y seguiremos con la explicación de la metodología que aplicaremos y el desarrollo del mismo. Para finalizar expondremos los resultados y conclusiones y las líneas de trabajo futuro con el que se podría continuar el desarrollo.

[1] https://aseguradorassanidad.wordpress.com/tag/tiempo-medio-espera/

2. Objetivos del proyecto

Podemos dividir los objetivos del proyecto en dos grupos: Los objetivos funcionales y los objetivos de diseño. Mientras que los primeros explicitan el funcionamiento de la aplicación y por ende, lo que el producto va a ofrecer a los clientes, los segundos determinan los aspectos técnicos a cumplir. Los objetivos funcionales son:

- Implementar una solución para ofrecer un portal con el que gestionar el proceso de consulta médica.
- Ofrecer a las administraciones (centros sanitarios y consultas privadas) un sistema con el que incorporar asistencia sanitaria de forma telemática a sus servicios.
- Ofrecer a los facultativos un conjunto de herramientas para gestionar las citas y que puedan realizar consultas por video conferencia además de manejar la información de los pacientes.
- Ofrecer a los pacientes una plataforma con la que podrán gestionar el proceso completo de consulta médica: solicitar una cita y recibir atención, además de poder consultar la información generada por las mismas.

Los objetivos de diseño:

- Utilizar un proceso de ingeniería de software bien definido.
- Definir una arquitectura que sea fácilmente integrable con cualquier sistema de gestión de pacientes.
- Definir los requerimientos del sistema y de los distintos perfiles de usuarios que harán uso del sistema.
- Implementar un modelo de datos relacional sin ambigüedades.
- Desarrollar la lógica de negocio del sistema.
- Diseñar e implementar un sistema de comunicación de video y audio para la interacción médico-paciente con un bajo coste de rendimiento y sin la necesidad de instalar software adicional.
- Diseñar y desarrollar un prototipo funcional para el acceso y manejo de la información.
- Elaborar la documentación necesaria.

3. Estado actual del tema

Como hemos comentado en la introducción, en los últimos años se ha producido un auge en el uso de la telemedicina para el tratamiento de enfermedades de carácter leve y no urgente. A continuación y con el objetivo de realizar una propuesta de valor que incorpore características diferenciadoras y añada a nuestro producto valores adicionales, realizaremos un breve análisis de las aplicaciones de telemedicina actuales más utilizadas y que nos han parecido más interesantes.

Teladoc (http://www.teladoc.com/)

Imagen 1: Logo de Teladoc (www.teladoc.com)

Empezó su actividad en 2002 con el objetivo de tratar dolencias típicas y no urgentes de sus usuarios: alergias, resfriados, infecciones, etc. Actualmente es la número 1 en Estados Unidos con 7.5 millones de usuarios registrados. Ofrece acceso a asistencia sanitaria por parte de personal cualificado los 365 días del año, las 24 horas al día y en el estado en el que se encuentre el paciente. La asistencia puede ser por teléfono o via web, a través de un ordenador, tablet o Smartphone.

El paciente debe darse de alta y rellenar un formulario con información básica general y su historial clínico: alergias, problemas de salud, hábitos de vida e historial médico familiar. Para realizar una consulta el usuario accede a una página en la que selecciona el método por el que desea contactar, teléfono o web, y el día en el que quiere que la consulta se realice; disponiendo de la opción de añadir un recordatorio por mail o mensaje de texto. Una vez establecidos los detalles de la consulta, debe seleccionar el método de pago. Para acceder a la consulta vía web al usuario se le habilita un botón en su perfil con el que acceder a la misma en el momento de la cita. El tiempo medio de respuesta es de unos 16 minutos.

Los médicos son todos titulados por la American Board Association y pueden preescribir medicamentos de sustancias no controladas y enviarlas mediante receta electrónica a la farmacia que el paciente solicite. Las especialidades de los mismos son la de medicina familiar y pediatría.

El sistema cumplimenta el estándar HIPAA, Health Insurance Portability and Accountability Act o Ley de Responsabilidad y Portabilidad del Seguro de la Salud[2] para la protección de los datos del usuario. Además poseen la certificación ISO 27002

[2] http://www.hhs.gov/ocr/privacy/hipaa/understanding/

para la seguridad de la información manejada[3]. Ofrecen aplicaciones nativas para Android e IOS.

Según su web tienen un 95% de satisfacción del usuario final y un 90% de consultas realizadas satisfactoriamente.

El modelo de negocio que plantean es el cobro de una subscripción anual al servicio con un coste de 150$. Una vez pagada la subscripción, el precio por consulta es de 36$.

MDLive (https://mdlive.com/)

Imagen 2: Logo de MDLive (www.mdlive.com)

Es la segunda compañia más grande de Estados Unidos dedicada a la telemedicina, aunque fue la que experimentó un mayor crecimiento en el 2014, por lo que se espera que este año desbanque a Teladoc, su máximo competidor. Se fundó en 2009 y su filosofía es similar a la de su rival, proporcionando teleconsultas con especialistas homologados durante todo el año mediante mail, videoconferencia o llamada telefónica.

El paciente se registra y rellena sus datos personales y su historial médico. Para realizar una consulta el paciente selecciona el tipo de facultativo al que quiere realizarla de entre los ofertados: pediatras, terapeutas o psicólogos familiares. Después elige de un listado de médicos disponibles el especialista, además del día y hora en el que quiere realizar la consulta. En la siguiente pantalla describe sus sintomas y el método por el que quiere que se contacte. En el momento de la consulta el paciente recibe un mensaje de texto y un correo, con el enlace a la videoconferencia (en caso de haber seleccionado dicha forma de consulta).

Todos sus médicos son titulados y pueden preescribir recetas de sustancias no controladas y enviarlas a la farmacia selccionada por el paciente. También garantizan la confidencialidad de los datos mediante las certificaciones Hipaa e ISO 27002. Utilizan como plataforma para la videoconferencia Vsee[4] y disponen de aplicaciones nativas para IOS y Android.

El servicio tiene un coste de 45$ por consulta, sin modelo de subscripción. También ofertan planes mensuales. Publicitan en su web que tienen un 94% de consultas realizadas satisfactoriamente.

[3] http://www.iso.org/iso/catalogue_detail?csnumber=54533/
[4] http://vsee.com/

American Well (https://www.americanwell.com/)

Imagen 3: Logo de American Well (www.americanwell.com)

Empezaron en el 2006 construyendo portales de telemedicina para hospitales pero su solución inicial era cara y costosa de mantener, por lo que decidieron pivotar y ofrecer su propio servicio online.

A diferencia de las anteriores sólo ofrecen videoconferencias a través de la web o dispositivos móviles. Para poder realizar una consulta se debe completar un perfil de usuario. Una vez tengamos nuestro perfil en la web, seleccionamos el tipo de consulta y el facultativo de entre una lista con los profesionales de la zona disponibles. Describimos nuestros síntomas y seleccionamos la forma de pago: la aplicación nos conectará con el doctor tan pronto como esté disponible. Entre las especialidades que ofrece la aplicación encontramos medicina general, nutricionistas y terapeutas. Como en el resto, todos los médicos poseen el título expedido por el organismo oficial estadounidense y pueden recetar sustancias no controladas.

Disponen de aplicaciones nativas para IOS y Android. Utilizan Vsee como plataforma de videoconferencia y el sistema cumple con la HIPAA.

El pago se hace por consulta de 10 minutos con un coste de 49.95$.

Doctor On Demand Well (http://www.doctorondemand.com/)

Imagen 4: Logo de Doctor On Demand (www.doctorondemand.com)

Es una de las compañias más recientes en ofrecer el servicio de consultas online ya que se fundó en el 2013. No opera en todos los estados y por el momento no ofrece disponibilidad las 24 horas del día. Entre sus servicios ofertados nos encontramos con medicina general y pediatría, piscología y un gabinete especial de lactancia.

El proceso es un poco más sencillo y directo que el de las anteriores aplicaciones: nos registramos e introducimos la información de pago. Seleccionamos el tipo de consulta, describimos nuestros síntomas, medicamentos que estamos tomando y alergias y realizamos la llamada de consulta. El sistema nos pondrá en contacto con el primer profesional disponible. Despues de la llamada, se puede añadir el doctor a una lista de favoritos, lo que aumenta las probabilidades de ser atentido por dicho doctor en las siguientes consultas o realizar llamadas directas si está disponible. Todos sus facultativos son titulados y pueden preescribir recetas de sustancias no controladas.

La plataforma de video que utilizan es AddLive, basada en tecnología WebRTC. Recientemente fue comprada por Snapchat y no admiten nuevos clientes, concentrando su labor en dar soporte a los que tenían anteriormente. Como el resto, dispone de aplicaciones nativas para Andorid e IOs y cumplen con el estándar HIPAA.

El coste es por tiempo de consulta y depende del especialista. El sistema da la posibilidad de ampliar el tiempo de consulta cuando éste llega al final. Los precios son:

- Medicina general:
  - 40$ por 15 minutos de consulta.
- Psicología:
  - 50$ por 25 minutos de consulta.
  - 90$ por 50 minutos de consulta.
- Consulta de lactancia:
  - 40$ por 25 minutos de consulta.
  - 70$ por 50 minutos de consulta.

Comparativa

En la tabla 1, mostramos una comparativa de los elementos más significativos de los distintos servicios estudiados:

Nombre	Coste	Servicios	Cliente final	Plataforma de video
Teladoc	- Subscripción: 150$. - Consulta: 36$	- Medicina general - Pediatría	Paciente	-
MDLive	- Consulta: 45$	- Medicina general - Pediatría - Psicología	Paciente	VSee
American Well	- Consulta 10 minutos: 49,5$	- Medicina general - Nutricionista - Psicología	Paciente	VSee

DrOnDemand	- Medicina general:
	  ○ 40$ por 15 minutos de consulta
	- Psicología:
	  ○ 50$ por 25 minutos de consulta
	  ○ 90$ por 50 minutos de consulta
	- Consulta de lactancia:
	  ○ 40$ por 25 minutos de consulta
	  ○ 70$ por 50 minutos de consulta
		- Medicina general
		- Psicología
		- Lactancia
			Paciente	AddLive

Tabla 1: Comparativa aplicaciones de telemedicina

4. Metodología

Para la realización del presente proyecto adaptaremos a nuestras necesidades la metodología Scrum. Scrum [S95] es un modelo de desarrollo ágil caracterizado por definir una estrategia iterativa e incremental. Iterativa porque divide el proceso de desarrollo del producto en iteraciones (o sprints) de varias semanas de duración e incremental porque en cada iteración se evoluciona el producto, ya sea implementando una nueva funcionalidad, corrigiendo algún defecto o modificando alguna característica ya existente. A la finalización de cada iteración es conveniente haber desarrollado un entregable (incremento) en forma de documento de análisis, prototipo a validar o funcionalidad completa; algo tangible que el cliente pueda evaluar y emitir un juicio.

Scrum se enmarca dentro del grupo de metodologías ágiles [M02] ya que al estar dividida en ciclos de desarrollos no muy grandes, nos permite presentar en un corto espacio de tiempo un prototipo funcional al cliente, pudiendo reaccionar en etapas tempranas a cambios en los requisitos y dotando de flexibilidad al proceso de desarrollo de software.

Los elementos de la metodología que adoptaremos para nuestro proyecto serán:

- Roles:
  - Product owner: Es el responsable de determinar la funcionalidad y las prioridades del sistema. En nuestro proyecto será el experto Daniel Gonzalez Santana, médico del Centro Hospitalario Universitario Materno-Infantil.
  - Equipo de desarrollo: Serán los encargados de desarrollar la parte técnica del producto. Está compuesto por el alumno y el tutor de proyecto.

- Artefactos:
  - Product backlog: Es el resultado de las primeras reuniones de análisis con el product owner y se corresponde con los requisitos a cumplir y las funcionalidades a implementar de todo el sistema.
  - Sprint backlog: Es el subconjunto de requisitos bien definidos del product backlog que se van a abordar en el sprint actual.
  - Incremento: entregable de la iteración. Puede ser un documento o un prototipo funcional. El product owner debe encargarse de verificar el incremento y ver que cumple con sus expectativas.

- Eventos:
  - Planificación del sprint: En esta reunión el product owner explica las prioridades y se seleccionan elementos del product backlog para añadirlos al sprint backlog en función de las estimaciones del equipo de desarrollo.
  - Retrospectiva: Después de cada iteración el equipo debe analizar el sprint e identificar las fortalezas y debilidades del equipo para aplicar cambios o ajustes a las sucesivas iteraciones.

- Revisión del sprint: Una vez finalizado el sprint se convoca una reunión con el product owner para presentar el trabajo realizado. Es conveniente que en esta etapa se presente el incremento y que éste sea algo tangible para los usuarios, como un documento o un prototipo. Además se presentará la próxima iteración.
- Sprint: Ejecución de la iteración. Es recomendable que su duración no sea muy elevada, entre 3 y 4 semanas.

En la figura 1 mostramos el ciclo de vida utilizado en el desarrollo del proyecto:

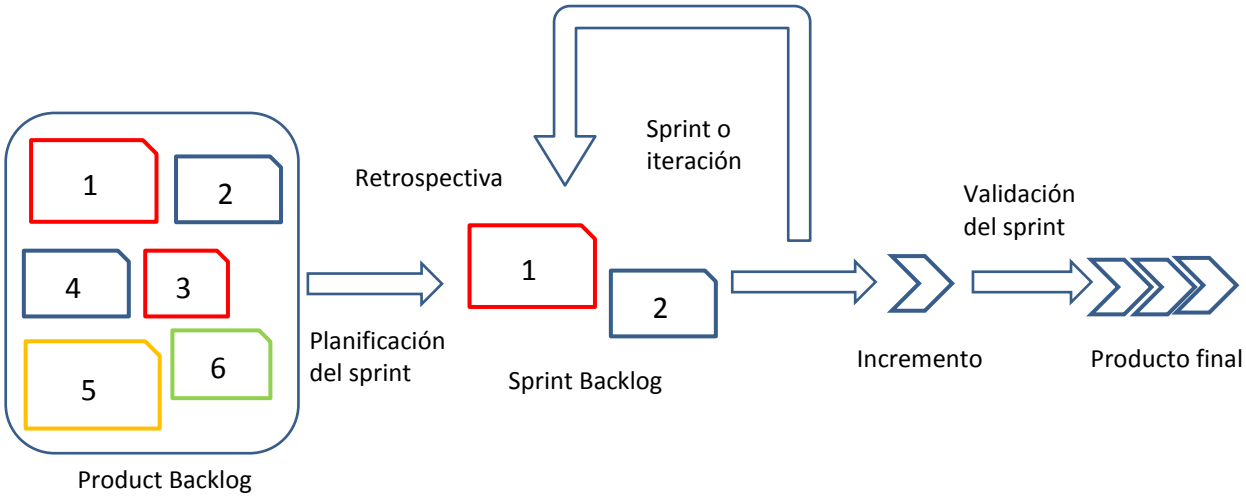


Figura 1: Ciclo de vida de la metodología

En cada uno de los sprints en los que está dividido el ciclo de desarrollo podremos trabajar en algúno de los siguientes aspectos de nuestro proyecto: el análisis funcional, el diseño, la implementación y/o la validación.

Durante el análisis funcional estudiaremos las tareas introducidas en el sprint backlog y desarrollaremos un documento que será presentado al product owner para gestionar que el proceso de las siguientes fases se ajuste a sus expectativas. Transformaremos las tareas del sprint backlog actual en historias de usuario. Una historia de usuario es una descripción breve de un requisito que debe cumplir nuestra aplicación. Se pueden representar de múltiples formas, nosotros utilizaremos una tabla con los siguientes elementos:

- Identificador: identificador de la historia.
- Identificador del product backlog: relación con los requisitos definidos en el product backlog.
- Título: breve texto identificativo.
- Descripción: texto explicativo. Debe ser lo más breve posible y con la estructura como… quiero… para… identificando a quien aporta valor, que se quiere conseguir y porqué.
- Dependencias: identificadores de las historias de usuario que deben haberse completado antes de poder empezar con la que nos ocupa.

Junto con las historias de usuario, desarrollaremos wireframes de las pantallas a implementar haciendo uso de alguna herramienta de mockup. Tanto las historias de

usuario como la realización de wireframes tienen como objetivo desarrollar un documento que el product owner pueda verificar y que permita detectar en etapas tempranas divergencias entre lo expresado por él mismo y lo entendido por el equipo de desarrollo.

En el diseño plantearemos una solución técnica para dar respuesta a los requisitos detectados en la etapa de análisis, haciendo uso de diagramas UML para representar el dominio del problema. Con diagramas entidad-relación representaremos los objetos a modelar mientras que los diagramas de secuencia nos darán una idea de las acciones que intervienen en los procesos. Por último, los diagramas de clases nos proporcionaran una visión cercana de las funciones y procedimientos a desarrollar para implementar la funcionalidad analizada.

En la etapa de implementación desarrollaremos el producto en base al diseño técnico anterior, teniendo como objetivo la creación de un prototipo funcional que cumpla con los requisitos de la etapa de análisis.

Por último, en la fase de validación, nuestro product owner comprobará que el entregable (incremento) se ajusta a sus necesidades. Es una fase de control de expectativas donde el propietario podrá proponer nuevos requisitos, modificar los actuales o cambiar la prioridad de los ya existentes.

La elección de esta metodología de desarrollo y la adaptación a nuestro proyecto da respuesta a las siguientes necesidades del mismo:

- Flexibilidad ante cambios: el cliente va contemplando la evolución del proyecto y tiene la posibilidad de introducir nuevos cambios en base a nuevas necesidades.
- Capacidad para tener una versión funcional de la aplicación en etapas tempranas.
- Reducción de riesgos al gestionar las expectativas del cliente en cada iteración y poder reaccionar ante ellos.
- Equipo de desarrollo pequeño.
- Alta disponibilidad de nuestro product owner.

Resumiendo el proceso de desarrollo expuesto:

- El product owner creará una lista priorizada con las características deseables de nuestro proyecto (product backlog).
- En base a las prioridades y la duración del sprint, el equipo junto con el product owner planificará el sprint actual, seleccionando el conjunto de tareas que van a ser abordadas (sprint backlog).
- Durante el sprint, el equipo se centrará en la realización de las tareas.
- Al finalizar el sprint. el trabajo realizado durante el mismo debe ser cuantificable ya sea en forma de documento, desarrollo, etc.
- El equipo realizará una reunión de retrospectiva para evaluar el sprint finalizado.
- Se realizará una reunión con el product owner para verificar el resultado del sprint.
- Se volverá al punto 2, se seleccionará otro conjunto de tareas y se planificará el siguiente sprint.

5. Plan de trabajo y temporización

Dividiremos en 4 etapas la duración total del proyecto:

- Etapa 1. Establecer las bases del proyecto: Realizar reuniones con el profesor y el product owner para definir el producto a implementar gestionando las expectativas y reduciendo las ambigüedades. (20 horas).

- Etapa 2. Definir el entorno tecnológico: Definir la arquitectura del sistema: elección de un Framework, de un ORM, sistema de gestión de base de datos, etc. Definir y eliminar riesgos tecnológicos. Instalar el entorno de programación así como las herramientas de edición de documentos. (80 horas).

- Etapa 3. Desarrollo de un prototipo funcional: Descomposición del desarrollo en iteraciones o sprints. Cada uno de estos sprints podrá tener una etapa de análisis, diseño, implementación y/o validación. (800 horas).

- Etapa 4. Documentación del Proyecto: Recopilar y generar la memoria del proyecto y realizar los manuales de usuario. (80 horas).

Duración total del PFC: 960 horas.

6. Definición del proyecto

Definiremos con la ayuda del product owner el producto a desarrollar. Las herramientas que utilizaremos serán las reuniones y entrevistas, obteniendo la pila del producto o product backlog. Una vez hayamos descrito el product backlog, éste deberá ser validado por nuestro product owner en un proceso cuya finalidad es la de gestionar las expectativas y comprobar que el cliente tiene una visión clara y sin ambigüedades, de lo que va a obtener con el producto final.

Nuestro objetivo es crear una plataforma que permita a las organizaciones que ofrecen servicios sanitarios, la configuración de un portal con el que poder dar asistencia por videoconferencia. Dicha herramienta comprenderá de un plan de citas para la consulta con el facultativo, la gestión de listas de espera para los citas, herramientas para la gestión de consultas y video-chat para la interacción médico-paciente.

El modelo de dominio del problema en el mundo real se muestra en la figura 2:

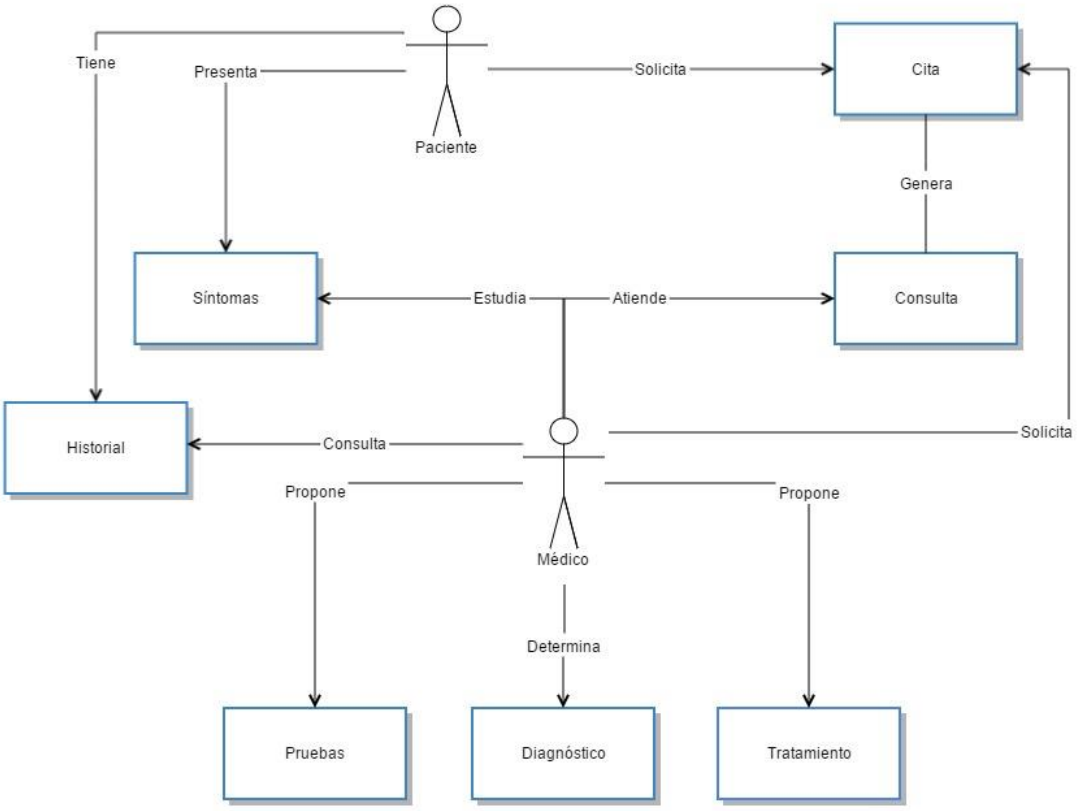


Figura 2: Modelo de dominio del problema

En el modelo anterior vemos como un paciente que presenta (o cree presentar) una serie de síntomas solicita una cita con su médico. Esta cita genera una consulta en un día y hora específicos. El médico atiende esa consulta y, una vez analizados los síntomas del paciente y consultado su historial, podrá determinar un diagnóstico y proponer un tratamiento para que el paciente se cure. Podrá además solicitar pruebas para determinar el diagnóstico y también concertar una nueva cita; bien para revisar el resultado de las pruebas del paciente o para ver la evolución del estado del mismo. En el modelo de dominio tenemos los siguientes conceptos:

- Paciente: persona que recibe los servicios de un médico u otro profesional de la salud y se somete a un examen o a un tratamiento.
- Médico o doctor: profesional que practica la medicina y que intenta mantener y recuperar la salud humana mediante el estudio, el diagnóstico y el tratamiento de la enfermedad o lesión del paciente.
- Consulta: proceso por el cual un médico atiende a un paciente en un tiempo determinado.
- Diagnóstico: procedimiento por el cual el médico identifica el estado de salud de un paciente.
- Cita: asignación de día, hora y lugar entre paciente y médico para que ocurra el proceso de consulta.
- Síntoma: referencia subjetiva que da un enfermo de la percepción que reconoce como anómala o causada por un estado patológico o una enfermedad.
- Prueba: exploración complementaria que solicita el médico para confirmar o descartar un diagnóstico.
- Historial (Historia clínica): documento donde se recoge la información necesaria para la correcta atención de los pacientes con valor informativo, científico y legal, en el que el médico deja constancia de su actuación.
- Tratamiento: Conjunto de medios que se emplean para curar o aliviar una enfermedad.

Una vez definido el modelo funcional en el mundo real de la aplicación que queremos construir debemos conformar la funcionalidad de la aplicación para cada usuario. Desde el punto de vista del paciente, éste debe poder acceder al sistema o darse de alta en el mismo. Una vez dentro podrá solicitar citas, consultar su historial, acceder a la sala de espera el día que tenga cita y recibir una consulta por video-chat. La funcionalidad deseada sería la siguiente:

- Rellenar formulario de alta.
- Logearse en el sistema.
- Pedir cita.
- Acceder a la sala de espera.
- Recibir consulta.
- Consultar historial.
- Obtener informes.

El médico debe poder acceder a la aplicación también. Consultar las citas, gestionar la sala de espera, ofrecer una consulta y consultar la información del paciente, así como poder generar y consultar diversos informes:

- Logearse en el sistema.

- Ofrecer consulta (Interacción con el paciente).
- Consultar historial del paciente.
- Gestionar citas.
- Consultar calendario de citas.
- Generar informes.
- Gestión de la sala de espera (debe poder seleccionar a un paciente de la sala, ver cuantos tiene en cola, etc.).

Aparte de los usuarios médico y paciente, debemos incorporar a nuestro proyecto un usuario cuyas funciones sean las de administrar el sistema:

- Logearse en el sistema.
- Gestionar pacientes.
- Gestionar establecimientos.
- Gestionar especialistas.
- Gestionar especialidades médicas.
- Gestionar pruebas médicas.
- Informes (datos globales-estadísticos).

En la tabla 2 configuramos el product backlog de nuestro proyecto:

Id	Prioridad	Descripción
1	1	El médico y el paciente deben poder realizar una consulta de audio y video.
2	1	Durante el proceso de consulta, el médico y el paciente deben poder comunicarse mediante chat.
3	3	Durante la consulta, el médico debe poder gestionar los datos de la misma.
4	2	El médico debe poder gestionar las citas que se encuentran en la sala de espera.
5	2	El paciente debe poder acceder a la sala de espera de la consulta del médico.
6	4	El médico podrá consultar la información del historial de los pacientes.
7	4	El paciente podrá consultar su información.
8	6	El médico podrá gestionar su calendario de citas.
9	7	El administrador podrá dar de alta nuevos usuarios, especialidades y pruebas.
10	8	El administrador podrá crear informes.
11	8	El médico tendrá acceso a informes.
12	9	El sistema debe permitir la forma digital.
13	9	El sistema debe ser sensible a la LOPD.
14	7	El paciente podrá darse de alta en el sistema mediante un formulario.
15	5	El paciente deberá logearse para acceder a la aplicación.
16	5	El médico deberá logearse para acceder a la aplicación.

Tabla 2: Product backlog

Una vez establecida las prioridades del sistema en base a las necesidades de nuestro product owner estableceremos un calendario de hitos del proyecto, añadiendo al product backlog desarrollado, las tareas que son implícitas a la implementación de nuestro proyecto:

- Establecer el entorno tecnológico del proyecto:
  - Objetivo: Determinar las necesidades del proyecto, las herramientas a utilizar, la arquitectura del sistema y eliminar los riesgos de desarrollo.
- Transmisión de audio, video y chat.
  - Objetivo: Transmitir audio, video y texto entre dos clientes web vía streaming.
- Consulta del paciente:
  - Objetivo: Poder atender una sesión de consulta con el especialista.
- Consulta del médico:
  - Objetivo: Poder atender una sesión de consulta con el paciente.
- Sala de espera:
  - Objetivo: Controlar el acceso de los pacientes a la consulta del médico.
- Historial del paciente:
  - Objetivo: Poder consultar la información del paciente por parte del médico.
- Información del paciente:
  - Objetivo: Poder consultar la información del paciente por parte de él mismo, pedir citas y obtener informes.
- Calendario:
  - Objetivo: Consultar el calendario de citas del médico.
- Administración del sistema:
  - Objetivo: Poder dar de alta usuarios (médicos y pacientes), especialidades, pruebas y establecimientos.
- Login de la aplicación:
  - Objetivo: Logearse en el sistema y gestión de permisos.
- Generar informes:
  - Objetivo: Generación de informes por parte del médico.
- Firma digital:
  - Objetivo: Poder firmar digitalmente los informes.
- LOPD:
  - Objetivo: Asegurarnos que la información guardada en la BBDD cumpla la LOPD.

6.1 Aplicaciones del sistema

El sistema comprenderá una solución completa al problema de la gestión de citas y consultas médicas con el valor adicional de que el proceso completo podrá realizarse a través de la web. El paciente podrá solicitar una cita, recibir la consulta desde casa y consultar su historial. Por otro lado, el sistema permitirá a los facultativos administar sus citas, gestionar la información de los pacientes y ofrecer consultas de forma centralizada, también favoreciéndose del proceso de interacción vía web. Además el sistema es fácilmente integrable con el sistema de historial clínico del cliente.

Desde el punto de vista funcional el sistema que desarrollaremos podrá cubrir un amplio abanico de necesidades. Algunas de las cuales podrían ser:

- Gestión de consultas: para hospitales o consultas médicas tanto públicas como privadas, los usuarios del sistema podrán gestionar todos los aspectos del proceso de consulta.

Imagen 5: Consulta médica

- Gestión de consultas de pacientes con un alto cargo contagioso: para la gestión de pacientes en condiciones de aislamiento.

Imagen 6: Médico trabajando con un traje de aislamiento

- Gestión de consultas en zonas remotas o de difícil acceso: lugares que son de difícil acceso y que requieran de una consulta rápida para evaluar el estado del paciente y la necesidad o no de trasladarse a la zona.

Imagen 7: Isla de Diómedes Menor

- Gestión de consultas en zonas desfavorecidas: lugares con pocos recursos económicos en los que no se puede mantener una consulta.

Imagen 8: Favela en Rio de Janeiro

6.1.1 Modelo de negocio

Hemos descrito anteriormente cuales son nuestros objetivos y algunos de los productos más populares relacionados con el servicio que pretende dar unuestra plataforma, por lo que pasaremos ahora a conformar nuestro modelo de negocio. Un modelo de negocio es básicamente el mecanismo por el que nuestro negocio va a generar beneficios. Para ello, utilizaremos la plantilla de modelo de negocio [O08] propuesta por Alexander Osterwalder que mostramos a continuación en la imagen 9:

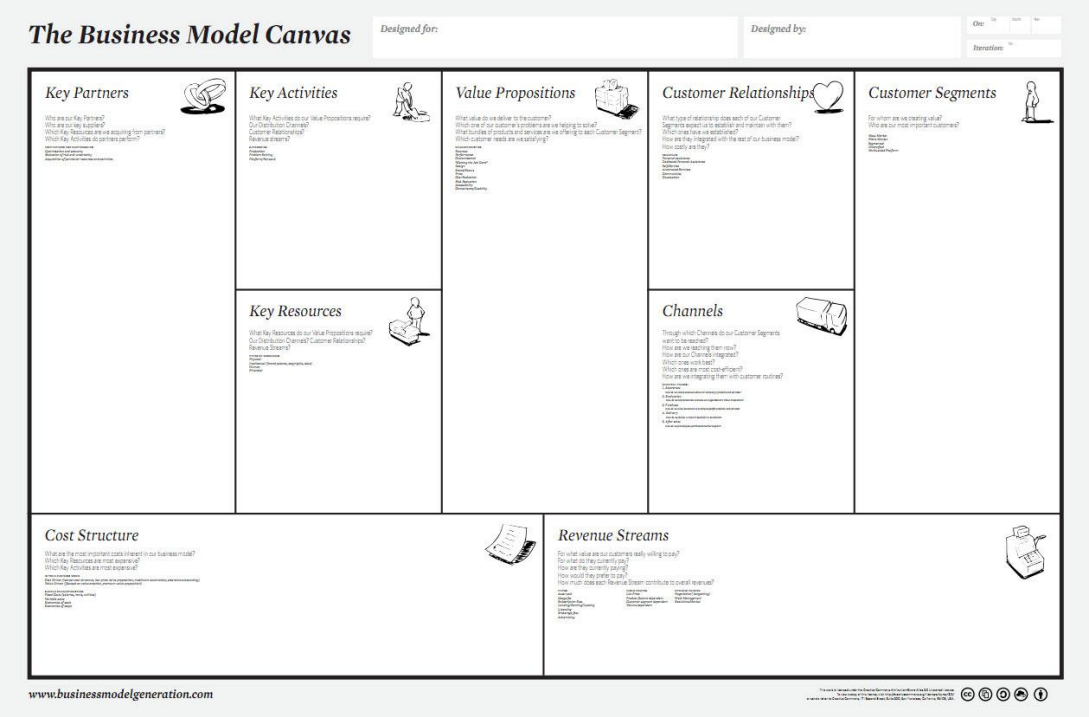


Imagen 9: Business Model Canvas

En dicha plantilla se intenta dar respuesta a los siguientes puntos: ¿Cuál es la propuesta de valor de nuestro producto? ¿A qué segmento de clientes va dirigido? ¿Por qué medios lo distribuiremos? ¿Qué relaciones estableceremos con los clientes? ¿Cómo ganaremos dinero con el producto? ¿Cuáles son nuestros recursos claves? ¿Y nuestras actividades? ¿Cuáles son nuestros socios clave? Y por último, ¿Cuál es la estructura de costes de nuesto proyecto?

La propuesta de valor es lo que diferencia nuestro producto de la competencia: ver que elementos innovadores introducimos para dar respuesta a las necesidades de nuestros clientes. Nuestro proyecto proporciona un portal web para la gestión de citas y consultas online para que los centros sanitarios oferten a sus pacientes asistencia telemática. Todo esto, con un coste de rendimiento relativamente bajo, permitiendo el acceso a la aplicación a cualquier equipo que disponga de un navegador con internet, facilitando la movilidad de los facultativos y centralizando el proceso. Además con el éxito del negocio, podríamos ofrecer la explotación de los datos generados por la aplicación.

Para ver si nuestro producto tiene mercado, debemos indentificar a que segmento de clientes va dirigida nuestra aplicación y estudiar sus necesidades para valorar nuestras oportunidades de negocio. Nuestros clientes potenciales serán los centros sanitarios y

las consultas privadas que desen ofrecer un portal web a sus clientes para la asistencia sanitaria por videoconferencia. Esta es la principal diferencia con los sistemas estudiados anteriormente ya que nuestro cliente objetivo no son los pacientes.

Debemos determinar que servicios aportaremos a nuestros clientes y que relaciones estableceremos con los mismos. Un servicio de soporte para la atención personalizada, formación para el uso de la aplicación y ayudas y manuales online es lo mínimo que deberíamos ofrecer. Además podríamos incorporar un servicio de integración de nuestro sistema con el sistema de historia clínica de los clientes.

Los canales de distribución determinan la forma en la que hacemos llegar nuestro producto a los clientes finales. En principio nuestro producto se distribuirá directamente por internet y también a través de comerciales.

Para monetizar el producto, nuestra fuente de ingresos principal vendrá de la venta de licencias. En el mercado actual, tan competitivo y con la crisis de fondo, pensamos que la mejor forma de generar ingresos es cobrar una pequeña licencia por facultativo y otra por paciente, además de un pequeño procentaje por cada consulta. De esta forma nuestro producto crecerá con la rentabilidad del comprador y el uso de la misma, en un esquema win-win.

Para generar un producto necesitamos de una serie de actividades estratégicas que den valor a nuestra propuesta. Éstas son el análisis, diseño e implementación de la plataforma, el soporte a la misma y la gestión de la distribución del producto.

Ademas de actividades, para lograr nuestro objetivo debemos especificar los recursos físicos, intelectuales, humanos y financieros que necesitaremos. Los recursos físicos serían un despacho y equipos para el desarrollo del producto, además del motor de videoconferencia. Los recursos intelectuales podrían ser la creación de una marca y la explotación de la base de datos del producto. Los recursos humanos serían los ingenieros de software para el desarrollo y mantenimiento del proyecto, los ITs para mantener la estructura del mismo, los técnicos de soporte para las relaciones con el cliente y el equipo de comerciales. Por último, como recurso financiero podríamos tener una línea de crédito para las primeras fases del proyecto, en las que no se generan ingresos.

Necesitamos también incorporar recursos externos a nuestro proyecto y crear asociaciones clave que cubran las necesidades que no podemos cubrir nosotros. Nuestros socios clave serían los proveedores de material informático, de Internet y de hosting. Además, para mejorar la aplicación necesitamos de socios expertos en el dominio de la información del problema, por lo que sería bien visto asociaciones con facultativos y administradores de servicios sanitarios.

Por último, necesitamos conocer la estructura de costos de nuestro proyecto para determinar la rentabilidad de la inversión del mismo. La inversión inicial debería cubrir los sueldos, la compra de los equipos y el alquiler de las instalaciones.

La figura 3 representa el modelo de negocio:

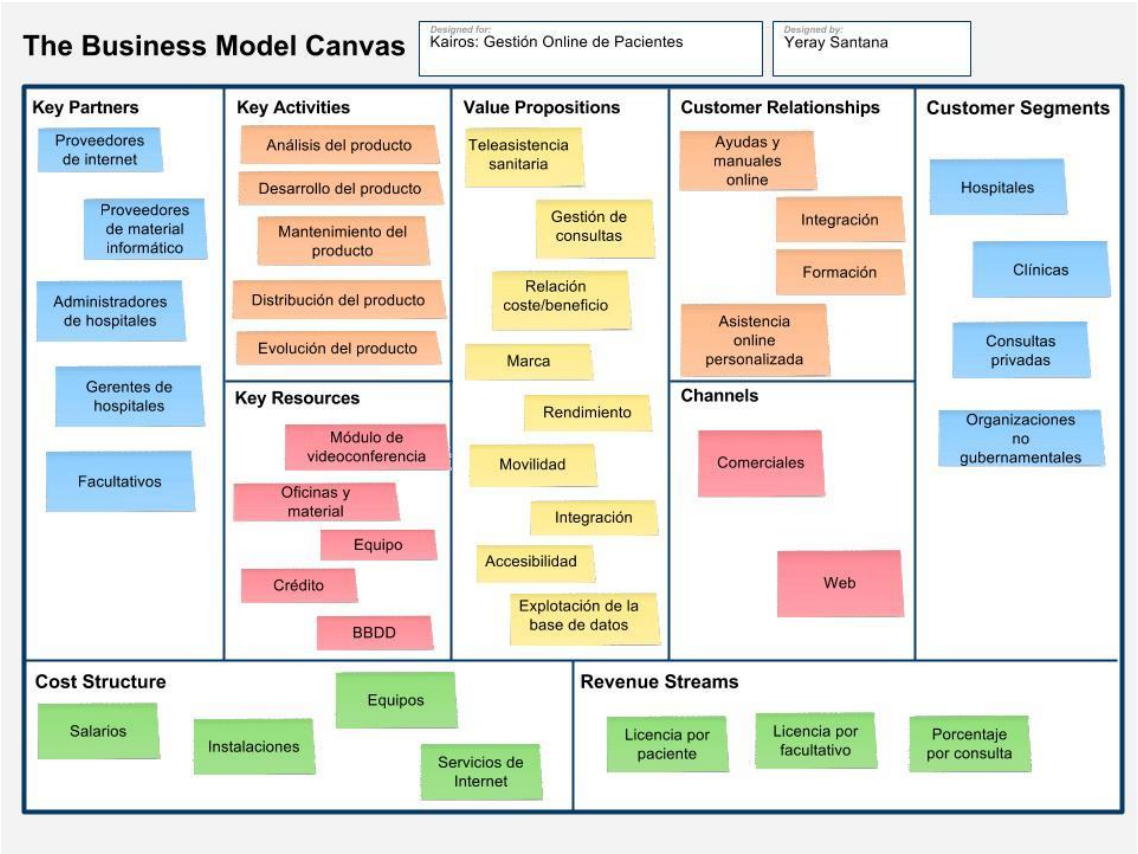


Figura 3: Modelo de negocio

7. Sprints

7.1 Sprint 1: Entorno tecnológico

En este primer sprint o ciclo de iteración estudiaremos las necesidades del proyecto para establecer las bases sobre las que desarrollaremos el producto. Los requisitos tecnológicos de un proyecto definen el entorno sobre el que se va a implementar la solución. En nuestro caso las características que definen dicho entorno son:

- Debe ser una aplicación web.
- Debe poder persistir la información (CRUD, Crear, Obtener, Actualizar y Borrar en BBDD).
- Debe poder transmitir audio y video de forma asíncrona entre dos clientes en distintas localizaciones.
- Debe poder implementar un sistema de paso de mensajes de forma asíncrona entre dos clientes en distintas localizaciones.

Es en este momento también cuando debemos tener en cuenta los riesgos tecnológicos que pueden hacer que el proyecto sea un éxito o por el contrario, un fracaso: la disponibilidad de la tecnología en el momento del desarrollo, el conocimiento de las áreas a tratar por parte del product owner, el conocimiento que posee el equipo de trabajo de las herramientas y lenguajes a utilizar, son los más determinantes. En esta etapa deberemos decidir si el proyecto se puede llevar o no a cabo y documentar las razones que nos han hecho tomar dicha decisión.

7.1.1 Definición del entorno tecnológico

Hemos decidido separar la implementación del proyecto en dos: uno con la funcionalidad para presentar y gestionar la información de la aplicación al que llamaremos KairosOMAC (Kairos Online Medical Assistance Client) y otro que utilizaremos para la gestión de la comunicación por video-chat y al que nombraremos KairosOVC (Kairos Online Video Chat). Al separar la capa de manejo de información de la de comunicaciones obtenemos dos sistemas complementarios y desacoplados sin apenas dependencias, permitiendo la integración de la implementación encargada de las comunicaciones de forma fácil y sencilla en sistemas de terceros.

Interfaz y gestión de datos

A pesar de que existen numerosos lenguajes para el desarrollo de productos web, cada uno con sus ventajas e inconvenientes hemos optado por elegir para la implementación de nuestra web PHP en su versión 5. El motivo principal que nos ha llevado a tomar esta elección es que PHP se ha mostrado a lo largo de los años como uno de los

lenguajes más extendidos para el desarrollo de aplicaciones web[5]. Además es un lenguaje sobre el que se tiene un elevado nivel de conocimiento por lo que reducimos uno de los componentes directos que afecta al riesgo del éxito o fracaso en el desarrollo de nuestro producto.

Para hacer uso de PHP utilizaremos un marco de trabajo que nos permita la automatización de código de las tareas sencillas y nos proporcione un esquema MVC (Modelo-Vista-Controlador). El framework elegido es Codeigniter. Codeigniter ofrece una solución de código abierto bastante completa para el desarrollo de sitios web dinámicos. Posee una amplia comunidad a sus espaldas y multitud de librerías que nos simplificarán el proceso de desarrollo de nuestro producto[6].

Para la interfaz utilizaremos HTML5 junto con la librería javascript JQuery para la presentación de información en la pantalla.

Base de datos

Para la persistencia de la información haremos uso de MySQL como sistema de gestión de base de datos. Podemos decir que las razones que nos han llevado a elegir este motor frente a otros son casi las mismas que nos llevaron a elegir PHP: amplia aceptación del sistema en el entorno de desarrollo de aplicaciones y elevado conocimiento del mismo por nuestra parte[7]. No obstante, aunque utilicemos MySQL, añadiremos al componente de persistencia un sistema de mapeo objeto-relacional u ORM[8]. El uso de un ORM añade una capa de abstracción a la base de datos, mediante el uso de objetos que contienen el código necesario para el acceso y manejo de datos. Este código es transparente para el usuario por lo que, desde el punto de vista del desarrollador, las funciones para la manipulación de los datos se hacen a través de objetos.

Las ventajas son fáciles de detectar: código más limpio, capacidad de reutilización del mismo (combinándolo con el patrón modelo de nuestro framework). Además, otra de las grandes ventajas de la utilización de un ORM es la posibilidad de cambiar la fuente de los datos sin tener que rehacer las consultas, lo que añade una característica interesante a nuestro proyecto: la capacidad de integración con cualquier otro sistema de gestión de pacientes independientemente del motor de base de datos que esté utilizando. Hemos decidido optar por Propel como ORM y los motivos se repiten: sistema extendido con una buena base de usuarios y documentación, además de elevado conocimiento del mismo por nuestra parte.

Servidor web

La elección de un servidor HTTP es bastante evidente; teniendo PHP y MySQL parece obvia la opción de utilizar Apache: multiplataforma, código abierto, ampliamente utilizado, comunidad de garantías, multitud de módulos que lo hacen altamente

[5] http://w3techs.com/
[6] http://www.php-developers.org/blog/codeigniter/advantages-and-features-of-codeigniter-framework.html
[7] http://db-engines.com/en/ranking
[8] http://www.orm-designer.com/article/orm-frameworks-for-php5-general-introduction

configurable, coste cero… son algunas de las características principales que han hecho de Apache uno de los servidores más utilizados en la red y por lo tanto, una excelente elección[9].

Sistema de gestión de consultas por video-chat

El siguiente aspecto tecnológico de nuestra aplicación es el de crear un sistema que permita la transmisión asíncrona de datos (audio, video y chat) entre distintos usuarios de la misma. Es un aspecto crítico del sistema ya que es sobre el que recae mayor riesgo tecnológico debido a la inexperiencia en el desarrollo de aplicaciones de este tipo y la dificultad de implementar un sistema que de soporte con calidad y garantías a las necesidades de comunicación de la aplicación.

En este punto también debemos diferenciar dos aspectos a tener en cuenta a la hora de implementar nuestro sistema de transmisión de audio y video: el cómo vamos a acceder a los recursos y el cómo vamos a transmitir los datos.

Para el acceso a los recursos del equipo: la cámara y el micrófono y la transmisión de este contenido haremos uso de la API WebRTC[10]. WebRTC es una interfaz de programación que provee de una serie de funciones y procedimientos para el acceso a recursos media a través del navegador y permitir comunicaciones en tiempo real entre distintos clientes (llamados peers) mediante Javascript y HTML5. Todo esto sin la necesidad de instalar plugins o librerías de terceros. El proyecto es relativamente nuevo ya que empezó en 2011 y está siendo desarrollado por la World Wide Web Consortium (W3C) lo que nos da una idea de la relevancia del mismo. Las necesidades principales que llevaron al desarrollo de la API han sido tres:

- Descargar, instalar y actualizar plugins en un proyecto puede ser una tarea tediosa.
- La utilización de plugins puede requerir de licencias y su integración a veces se hace demasiado compleja.
- Es muy común que el usuario sea reacio al uso o instalación de plugins.

El último problema tecnológico a solventar es el de proporcionar una infraestructura que de soporte a lo anterior. Al utilizar WebRTC, lo ideal sería utilizar un servidor javascript como es Node.js[11]. Node es un entorno Javascript de lado de servidor que utiliza un modelo asíncrono y dirigido por eventos y corre sobre el motor V8 desarrollado por Google.

Lo que hace que Node sea altamente recomendable para aplicaciones de este tipo[12] es la forma en la que gestiona sus hilos de ejecución. Node se programa bajo un único hilo (a diferencia por ejemplo de Apache, que crea un nuevo hilo por cada conexión cliente-servidor). Node sólo creará un nuevo hilo si existe una operación bloqueante, esto hace que el servidor pueda manejar un elevado número de conexiones. Además Node mantendrá siempre la conexión abierta con el cliente y enviará los datos cuando estén

[9] http://w3techs.com/
[10] http://www.webrtc.org/
[11] http://www.html5rocks.com/en/tutorials/webrtc/basics/
[12] http://www.rmunoz.net/introduccion-a-node-js.html

disponibles, sin necesidad de estar haciendo continuamente peticiones. Las características de nuestro proyecto que hacen que nos hayamos declinado por Node son:

- Tiempos de respuesta bajos.
- Alta concurrencia.
- Posibilidad de un elevado número de usuarios.
- Conexión persistente entre el navegador del cliente y el servidor.

Sistema de control de versiones

Una vez definido nuestro entorno tecnológico, se nos hace necesario el incorporar una herramienta que nos permita gestionar los cambios de código durante la implementación. Por ejemplo, que nos permitan revisar el histórico de desarrollo, recuperarnos ante la pérdida del código del proyecto, crear instantáneas del mismo o volver a alguna versión anterior. A este tipo de sistemas se les llama sistemas de control de versiones (SCV). Para nuestro proyecto utilizaremos Git como SCV y alojaremos el código en Bitbucket. Bitbucket es un servicio gratuito de alojamiento para sistemas basados en Git.

La estructura que utilizaremos para nuestro repositorio será la siguiente [PM08]:

- Master: rama principal del proyecto.
- Branch: rama de cada iteración.
- Tag: entregable después de cada iteración.

El proceso que seguiremos para trabajar con el repositorio será el sacar una rama para cada iteración. Al finalizar la misma enseñaremos al product owner el resultado de la iteración en la rama. Si se cumple con las expectativas del cliente, unificaremos el branck con la rama maestra (trunk) en un proceso que se llama merge, sacaremos un tag, que es una 'foto' del estado actual del proyecto y crearemos un nuevo branch para la siguiente iteración. La figura 4 muestra el proceso de versionado.

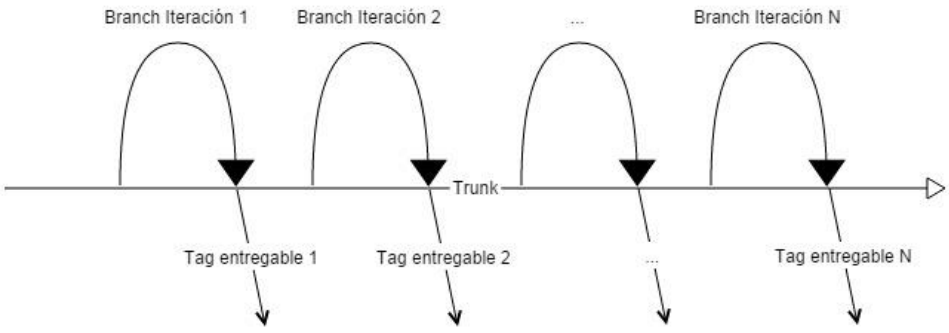


Figura 4: Proceso SVN

Otras herramientas

Además, como durante la realización del proyecto debemos completar documentos de análisis, diseño y realizar una memoria, utilizaremos las siguientes herramientas:

- Eclipse: como IDE para la implementación de código.

- Microsoft Word 2010, para la realización de los documentos.
- Gliffy Diagrams para los distintos diagramas.
- Mockflow para la realización de los wireframes.

7.1.2 Entorno tecnológico.

Como hemos descrito en los puntos anteriores tenemos las dos capas en las que hemos dividido nuestro entorno y que mostramos en la figura 5: la capa de la aplicación que se encarga del manejo de información, KairosOMAC, montada en un entorno Apache, y la capa encargada de las comunicaciones, KairosOVC, implementada sobre un entorno Node. Esta separación nos permitirá poder utilizar en otros proyectos cada una de las capas de forma independiente, con un bajo coste de integración.

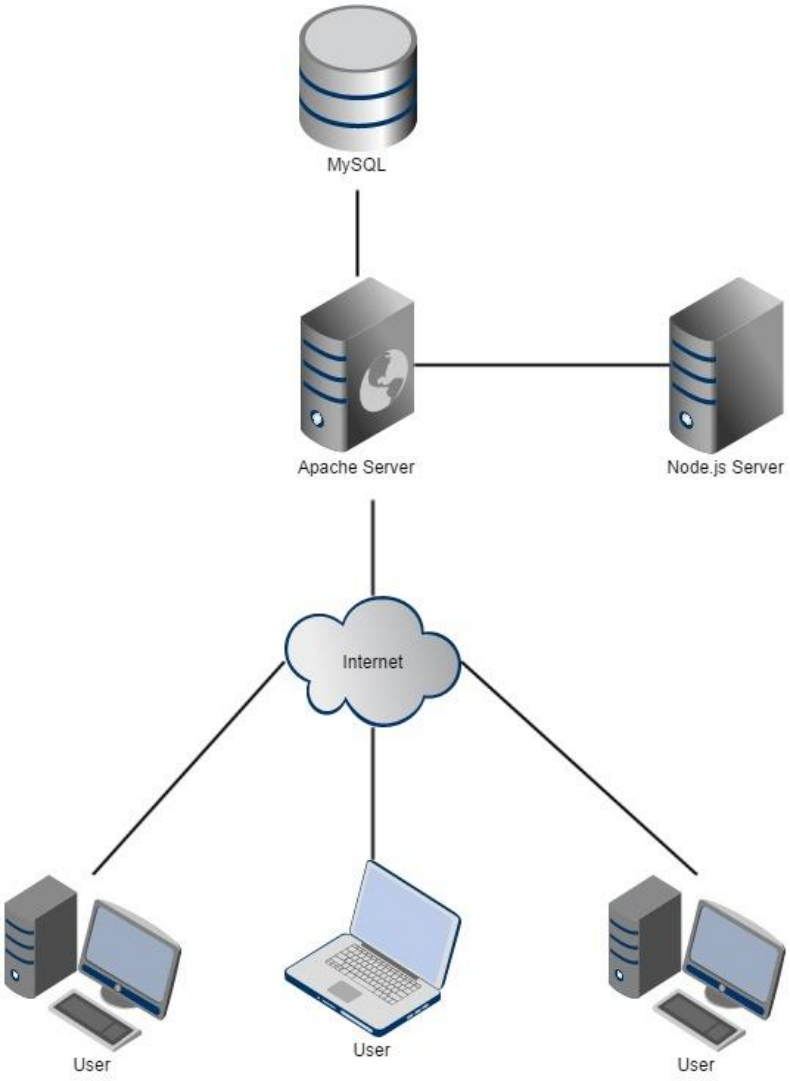


Figura 5: Entorno tecnológico

En la figura 6, mostramos la arquitectura de nuestro sistema de gestión de citas y consultas médicas: montado en un entorno Apache tenemos nuestro framework PHP Codeigniter para el desarrollo de la lógica de negocio de nuestra aplicación. Para la capa de presentación y experiencia de usuario utilizamos HTML5 y JQuery. Para el manejo de los datos tenemos Propel como ORM encargado de la gestión de los procesos CRUD, abstrayendo a nuestra aplicación de los detalles del motor de BBDD, que en nuestro caso es MySQL. Además utilizamos los siguientes plugins.

- JQuery UI Notification: Plugin JQuery para mostrar mensajes al usuario.

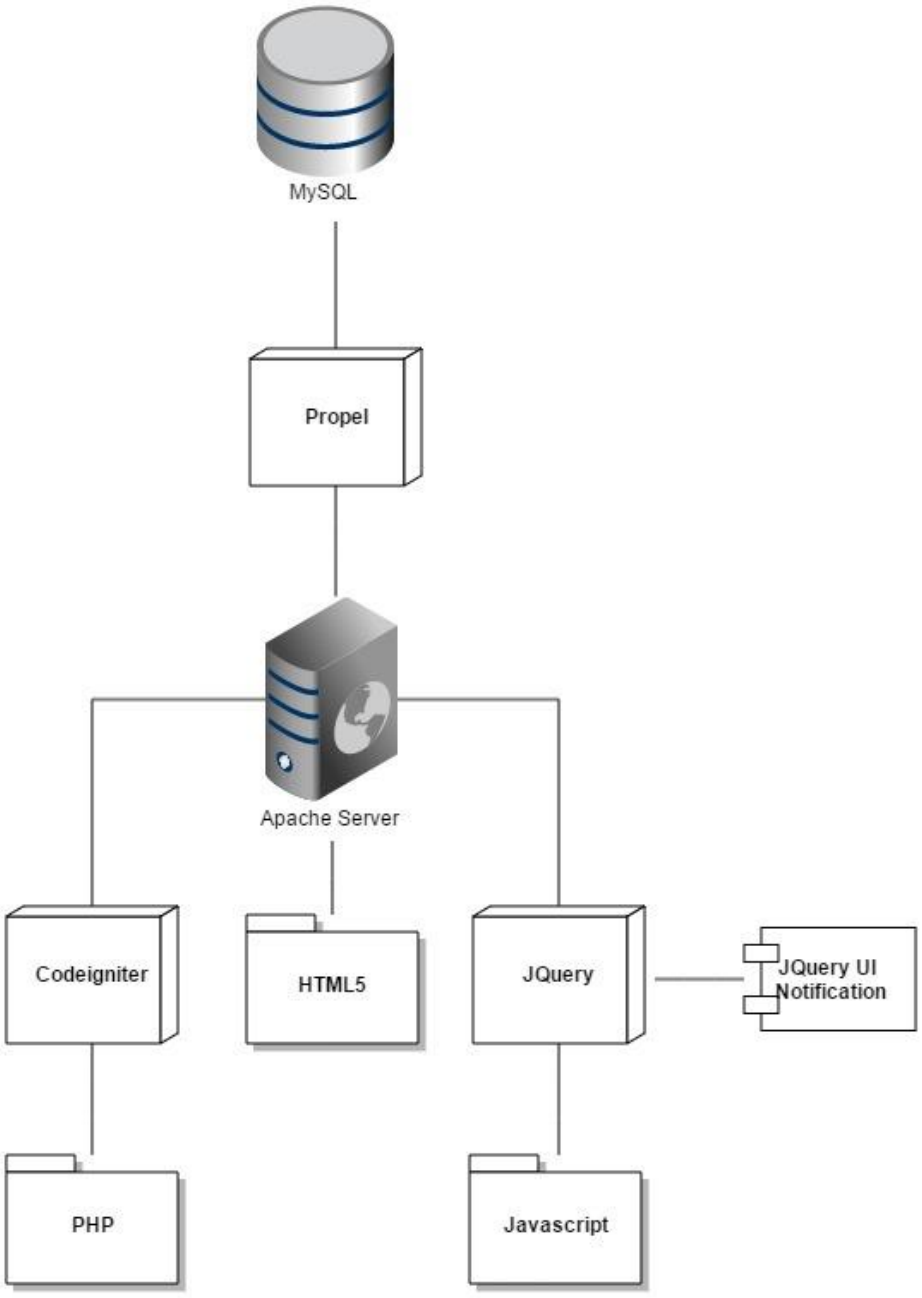


Figura 6: Capa de gestión de datos

La arquitectura del sistema de comunicación se muestra a continuación en la figura 7. Un servidor Node.js, que ejecuta código Javascript. Como hemos comentado esta separación lógica nos permitirá exportar o utilizar el servicio de comunicaciones en otros proyectos. Las librerías utilizadas:

- Node-static: módulo para Node.js que permite servir contenido estático.

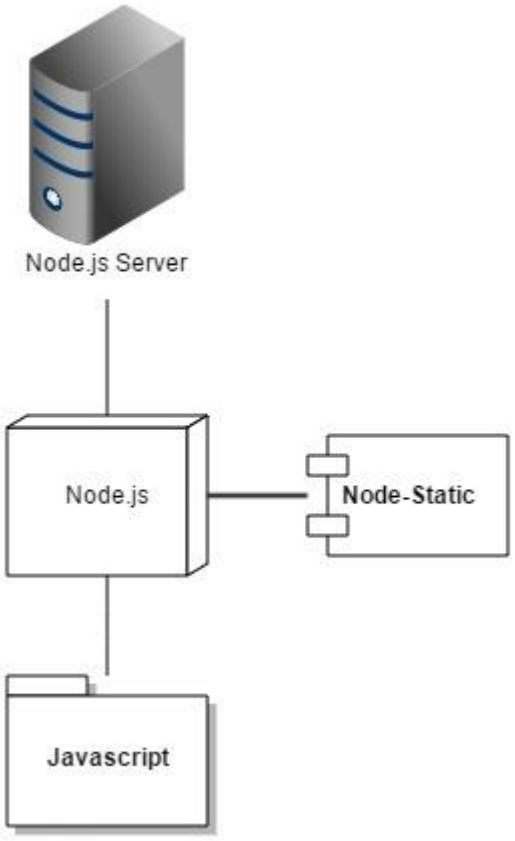


Figura 7: Capa de gestión de las comunicaciones

7.2 Sprint 2: Sistema de comunicación por video y audio

En esta segunda iteración del proyecto abordaremos una de las funcionalidades principales de nuestro sistema y la que conlleva mayor riesgo técnico. Como se ha comentado previamente, es fundamental abordar este tipo de hitos lo más temprano posible y evaluarlos para determinar los riesgos a los que se expone el proyecto.

El objetivo a cumplir en esta etapa es lograr una comunicación fluida entre dos clientes o peers a través de la web, con un bajo coste de rendimiento y sin la necesidad de plugins. Como hemos comentado en el apartado anterior, haremos uso de la API WebRTC para la comunicación en tiempo real a través del navegador web entre dos clientes. En el momento de la realización del proyecto los navegadores que soportaban la API eran:

- Navegadores de escritorio:
    - Google Chrome, a partir de la versión 23.
    - Mozilla Firefox a partir de la 22.
    - Opera a partir de la 12.

- Navegadores móviles:
    - Google Chrome 28 para Android.
    - Mozilla Firefox 24 para Android.
    - Opera Mobile 12 para Android.

Resaltar que cada navegador implementa la API de forma independiente. Los esfuerzos en la estandarización de la misma es uno de los objetivos de la W3C. Internet Explorer y Safari no desarrollan de forma nativa la API pero existen numerosos plugins que permiten al navegador soportar dicha funcionalidad, aunque se espera que la incorporen en próximas versiones. Además, y afortunadamente para nosotros, existen numerosos shims que nos permiten abtsraernos de las diferentes implementaciones, haciendo transparente la nomenclatura que cada navegador utiliza para la representación de la API.

Los tres pilares básicos sobre los que se sostiene WebRTC para lograr su propósito son:

- MediaStream: es una representación de un stream de audio y/o video, lo que permite al navegador su uso y manejo.
- PeerConnection: es el objeto que establece la comunicación directa entre clientes y transmite los elementos media.
- DataChannels: objeto que permiten a los navegadores compartir datos de carácter no media entre clientes.

Con todo esto, nuestra aplicación será capaz de ofrecer un sistema de videoconferencia con las siguientes características:

- Infraestructura tecnológica económica de mantener.
- Facilmente integrable en sistemas de terceros
- Bajo consumo de recursos.

- Sin necesidad de que el cliente realice la instalación de ningún plugin.
- Comunicación entre clientes independientemente de la forma en la que estén conectados a Internet (routers, firewalls, etc.).

7.2.1 Obtención de los elementos media

Empezaremos con la implementación de nuestra librería de comunicaciones. En esta primera etapa mostraremos como se puede capturar los elementos media del equipo a través del navegador usando el componente getUserMedia de WebRTC. La función[13] está diseñada para acceder a los streams de audio y video de los dispositivos media locales y proporcionar una interfaz para el manejo de los mismos. Esta función acepta tres parámetros:

- Un objeto con las restricciones: si queremos capturar audio y su bitrate, video y su resolución, etc.
- Una función o callback en caso de poder acceder a la funcionalidad con un objeto de tipo LocalMediaStream pasado como argumento, que representa el stream media capturado. El formato de dicho stream nos permitirá mostrarlo por pantalla (mediante un elemento HTML5 video, por ejemplo) o pasarlo como parámetro a un objeto RTCPeerConection para su posterior transmisión por la red a otro usuario.
- Una función o callback en caso de producirse un error al intentar acceder a la API.

En la imagen 10, mostramos el resultado de la captura de video mediante getUserMedia.

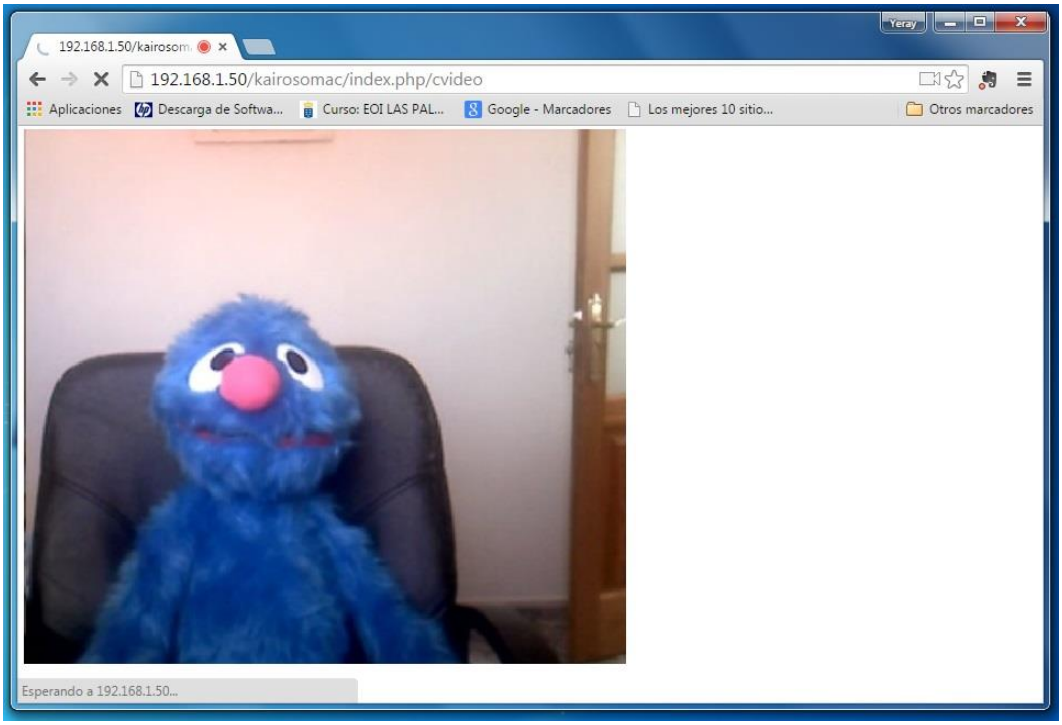


Imagen 10: Captura de imagen mediante getUserMedia

[13] http://w3c.github.io/mediacapture-main/getusermedia.html

7.2.2 Streaming de elementos media

El siguiente paso una vez capturado el video y el audio es transmitirlo desde un peer a otro peer haciendo uso de la interfaz RTCPeerConection[14]. Podemos descomponer el proceso de configuración de la transmisión en los siguientes pasos [M13]:

- Conectar a los usuarios: el primer paso es identificar a los usuarios y proporcionarles un acceso común a la funcionalidad.
- Control de la transmisión: Una vez conectados los dos peers, debemos proporcionar un mecanismo para que puedan enviarse mensajes de control para la configuración de la transmisión.
- Intercambio de información media a transmitir: cada cliente debe ponerse de acuerdo en el tipo y el formato de los elementos que se van a transmitir.
- Intercambio de información de red: una vez establecido el mecanismo de señalización los dos clientes deben empezar un proceso de descubrimiento de rutas de red, por las que podrán acceder entre ellos.
- Comenzar con el envío de los elementos haciendo uso el objeto RTCPeerConection.

Conectar a los usuarios.

Lo primero que debemos hacer es establecer un mecanismo para que los participantes se identifiquen. La forma más sencilla es que ambos clientes accedan a la misma página y se conecten a un servidor que los identifique y les proporcine los mecanismos necesarios para el control de la comunicación.

Control de la transmisión.

Una vez hemos puesto en contacto a los clientes, debemos establecer un mecanismo de control de la transmisión con el que puedan enviar y recibir mensajes. A este proceso se le denomina ‘señalización’. Los objetivos de dicho proceso son los siguientes:

- Obtener la información de los clientes como son la dirección IP y el puerto.
- Coordinar el proceso de comunicación para iniciar, cerrar la sesión y manejar los errores en dicho proceso.
- Gestionar las capacidades de cada sistema: codecs, resolución, etc.

El proceso de señalización no se contempla dentro de la especificación de la API, por lo que cada implementación es libre de utilizar los mecanismos que crea adecuados. Sin embargo, la implementación de un servidor de señalización debe seguir un enfoque orientado por el protocolo JSEP[15]. Además separar el servicio de señalización en una capa distinta en vez de tenerlo embebido en el navegador, nos evita problemas como pueden ser la pérdida del estado de la comunicación si por algún problema tuviéramos que recargar el navegador. En la figura 8 mostramos el esquema de señalización JSEP.

[14] http://w3c.github.io/webrtc-pc/#rtcpeerconnection-interface
[15] Javascript Session Establishment Protocol, http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03

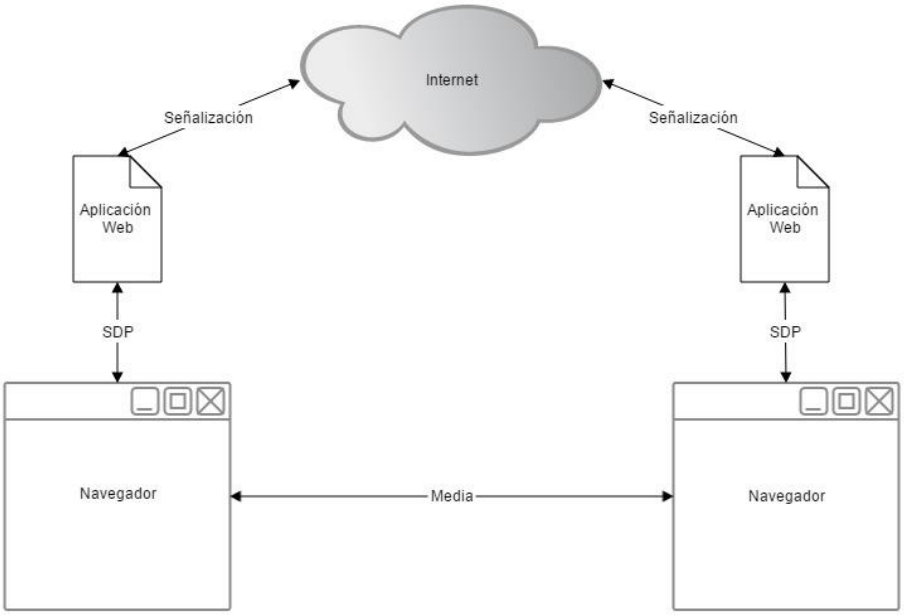


Figura 8: Modelo de señalización JSEP

Describamos ahora el blujo básico de interacción bajo este protocólo. Supongamos que dos clientes quieren entablar una comunicación. Uno de los clientes, el emisor, se conectará al servidor y realizará una petición para crear un canal con el que poder comunicarse. Si el canal no existe, como es el caso, el servidor lo creará y se lo notificará al emisor. El receptor, que quiere entablar una conversación con el emisor, se conectará al servidor y realizará una petición para crear el canal, pero como éste ya existe, solicitará el uso del mismo al servidor. Una vez el receptor ha ingresado en el canal, se enviará una notificación al emisor. Entonces el proceso de comunicación podrá comenzar, enviándose mensajes entre los dos clientes con la información necesaria para controlar el proceso de comunicación (descriptores de media e información de red). Recordemos que éste proceso es sólo para el control de la transmisión, no para la transmisión en si. Cualquiera de ellos podrá terminar la conversación desconectándose del servidor en cualquier momento. Ilustramos el proceso en la figura 9:

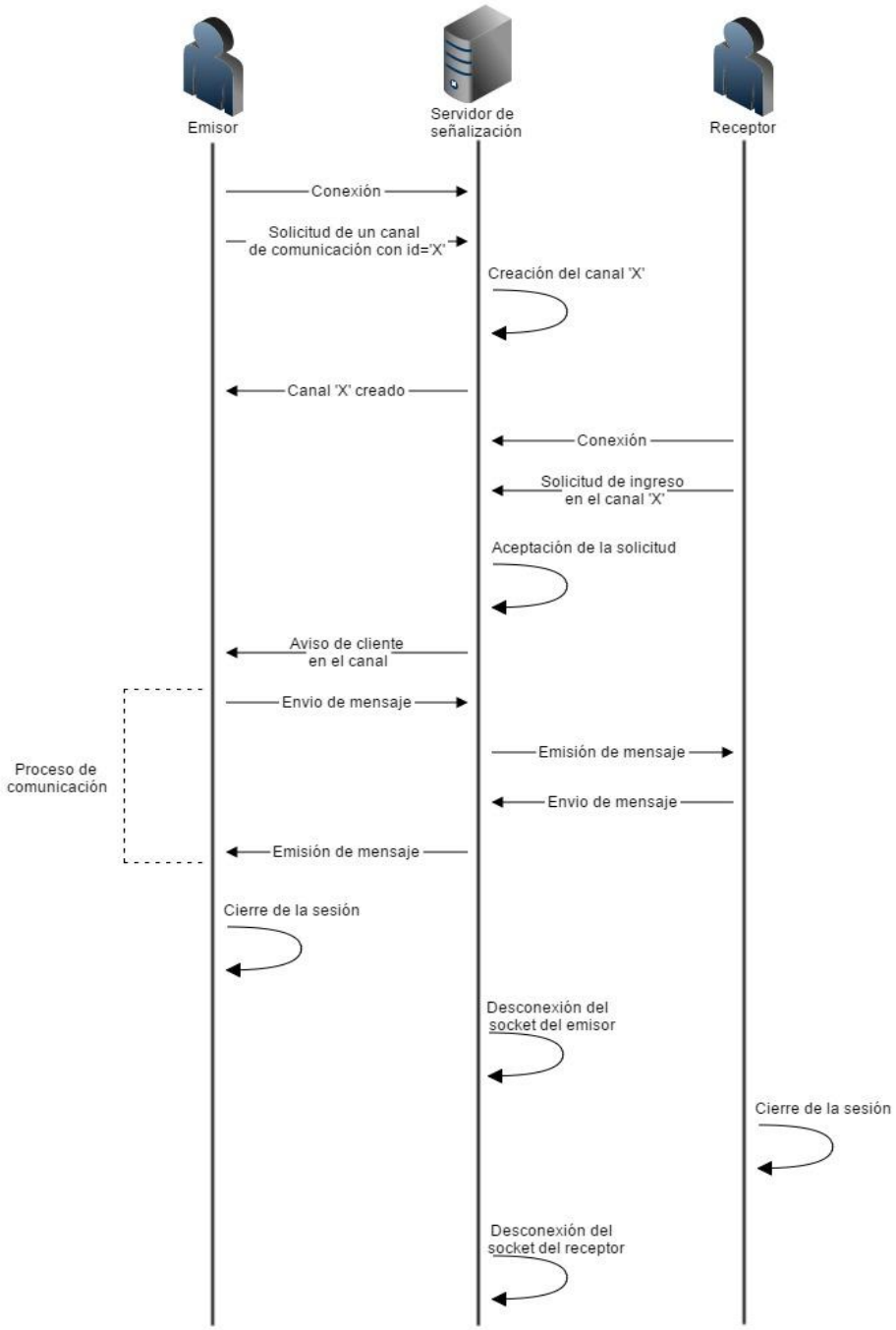


Figura 9: Proceso de señalización

Para la implementación del servidor de señalización hemos optado por utilizar websockets en nuestro entorno Node.js [LR14]. WebSocket[16] es una tecnología que proporciona un canal de comunicación bidireccional, asíncrona, persistente y full-duplex sobre un único socket TCP, evitando abrir múltiples puertos entre aplicaciones cliente-servidor: una vez establecida la conexión a través de un socket, ésta permanecerá activa hasta que uno de los lados decida cerrarla. Con esta API se pueden enviar mensajes a un servidor y recibir respuestas controladas por eventos, lo que evita al cliente tener que estar siempre consultando al servidor (polling) con el consumo de

[16] https://tools.ietf.org/html/rfc6455

recursos que esto implica. Para implementar esta tecnología en nuestro servidor Node haremos uso del paquete Socket.Io[17].

Intercambio de información de sesión

Una vez implementado nuestro servidor de señalización ya disponemos de un mecanismo para negociar el contenido media que se van a intercambiar los clientes. El modelo de señalización propuesto por JSEP y utilizado por nuestro servidor se basa en el intercambio de metadatos con formato SDP[18] (Session Description Protocol) entre los clientes, mediante el sistema de publicar una oferta y obtener una respuesta. Veamos un ejemplo del proceso de señalización bajo el modelo JSEP. Supongamos que un cliente emisor quiere realizar una llamada a otro cliente receptor. El emisor crea una oferta con la información de su SDP y la envía al receptor por medio del servidor de señalización. Cuando es recibida por el receptor, éste crea una respuesta a dicha oferta, la relaciona localmente para guardar la referencia y la envía al emisor. El emisor la recibe y la asocia con la oferta emitida. El proceso lo ilustramos en la figura 10:

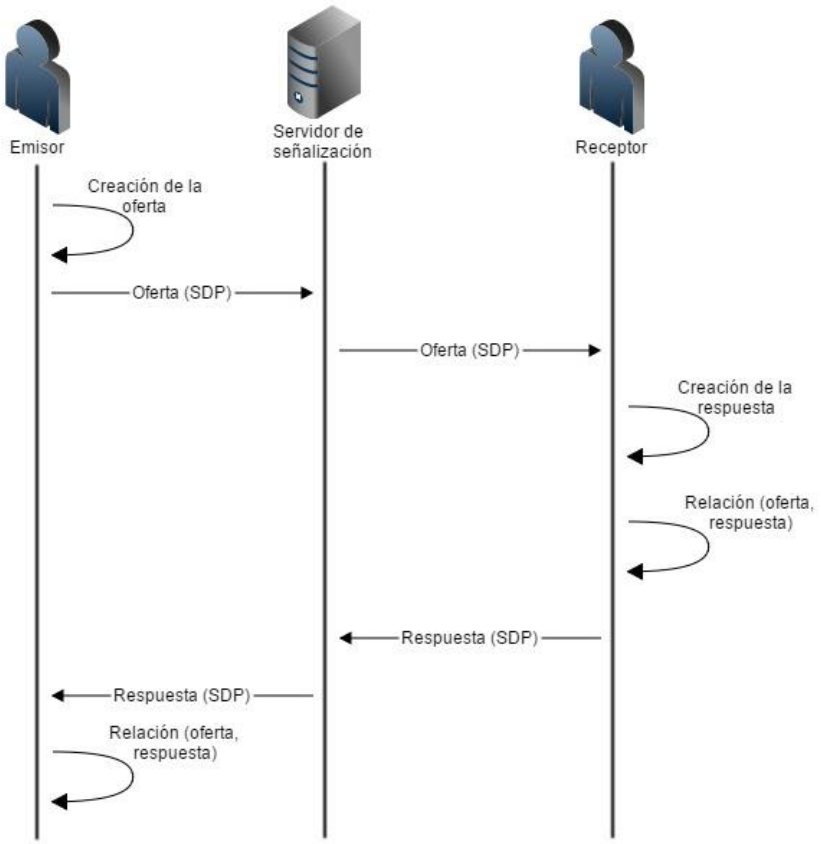


Figura 10: Intercambios de SDPs

SDP o Session Description Protocol es en protocolo que describe el formato de los parámetros para la participación en una conexión multimedia, como pueden ser el tipo

[17] http://socket.io/
[18] https://www.ietf.org/rfc/rfc2327.txt

de contenido, el formato, los codecs... o lo que es lo mismo, qué se va a transmitir y como se va a manejar el contenido media que se reciba. SDP no se encarga de la transmisión del contenido, sino de la negociación entre los clientes de las capacidades de cada uno.

Intercambio de información de red

Además del intercambio de información media durante el proceso de señalización, los participantes envían información de la dirección y del puerto donde pueden ser localizados: a este proceso se le conoce por el nombre de "búsqueda de candidatos". El objetivo de intercambiar esta información de red es el de poder transmitir directamente el contenido media de un cliente a otro cliente sin necesidad de elementos mediadores. Para este proceso de descubrir interfaces y puertos de red utilizaremos el framework ICE o Interactive Connectivity Establishment[19]. El objetivo de dicho framework es encontrar la mejor ruta entre los dos clientes solventando las complejidades de la topología de la red.

En un entorno real, como el que se muestra en la figura 11, los clientes no se conectan a la red directamente sino que lo hacen a través de algún dispositivo, como puede ser un router o un firewall. Cada cliente posee una dirección IP privada que lo identifica dentro de la red a la que pertenece y que no es accesible desde Internet. Para poder comunicarse con el exterior, se utiliza un mecanismo denominado NAT o Network Address Translation, para convertir las IPs privadas, que son únicas y accesibles desde Internet, en públicas y permitir a los equipos enviar peticiones al exterior y realizar el proceso inverso de convertir las direcciones públicas en privadas, para que el equipo que envió la petición, pueda recibir una respuesta.

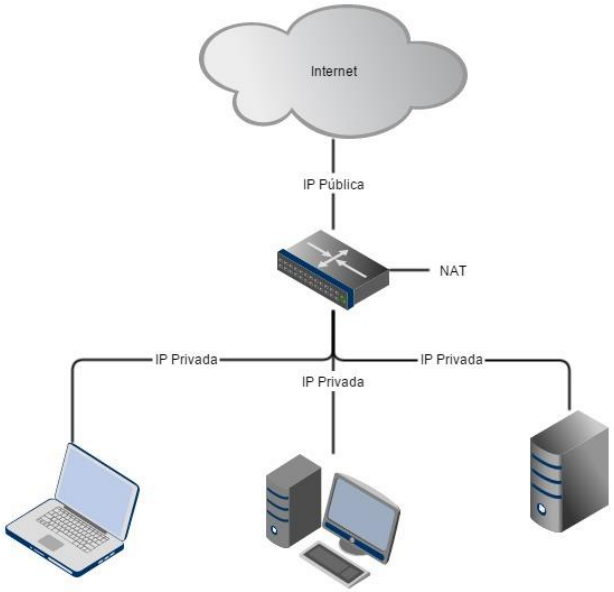


Figura 11: Topología de red NAT

[19] https://tools.ietf.org/html/rfc5245

El funcionamiento del framework es bastante sencillo y lo mostramos en la figura 12: Primero, tratará de realizar una conexión directa, si falla (como para el caso de equipos detrás de NATs), intentará obtener una dirección externa a través de un servidor STUN (Session Traversal Utilities for NAT). Dicho servidor se encarga de descubrir la dirección IP pública de los clientes y las reglas de NAT que siguen, enviando de vuelta esta información a cada cliente.

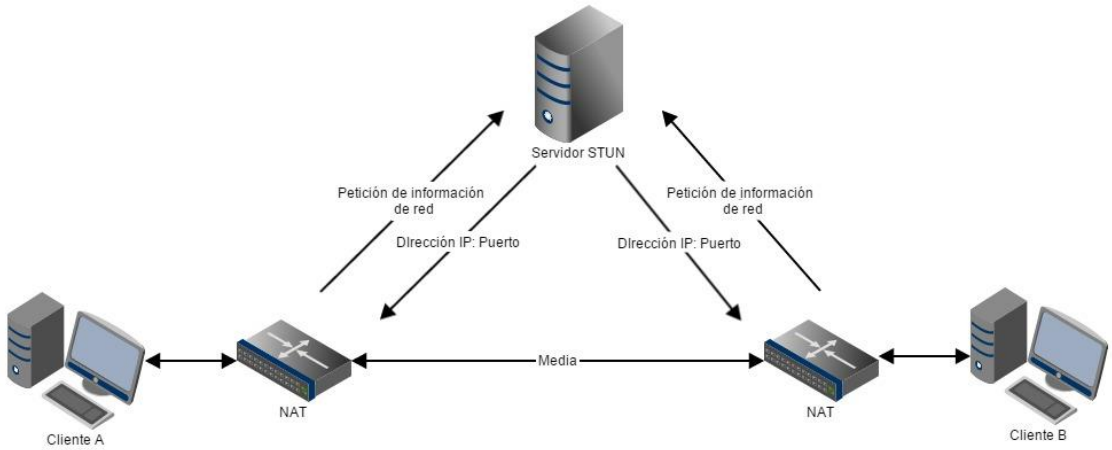


Figura 12: ICE mediante servidor STUN

Si el proceso falla, el framework tratará de conectar los clientes mediante un servidor TURN (Traversal Using Relays around NAT), que son servidores con direcciones públicas que actuarán de intermediarios en el proceso de transmisión del contenido media. En la figura 13 vemos el proceso:

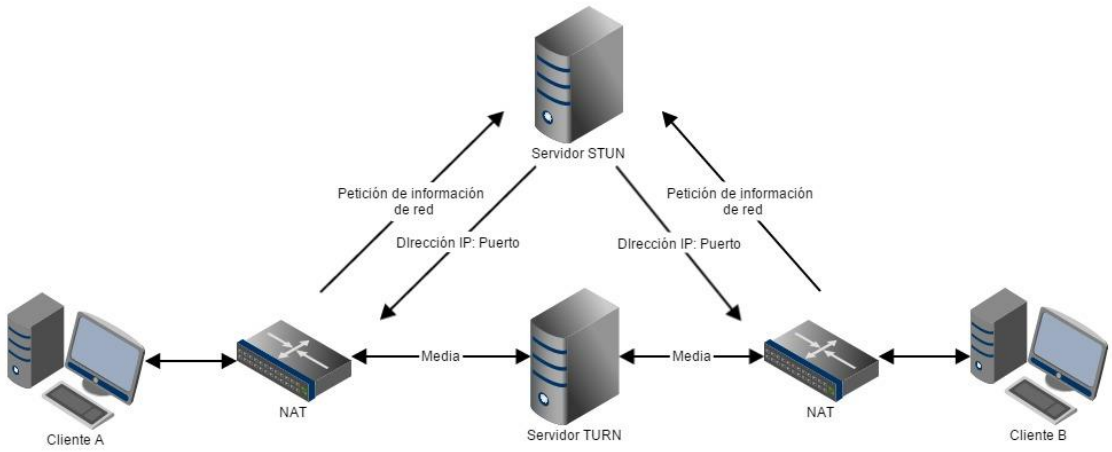


Figura 13: ICE mediante servidor TURN

Hacer notar que ICE se encarga solamente de la transmisión de los elementos media; no establece ningun elemento para el control de la transmisión (de esto se encarga el proceso de señalización). Una vez han finalizado los procesos de intercambio de información de red y de media, es cuando puede comenzar el streaming de elementos media.

Transmisión del contenido media

Con la implementación del servidor de señalización, tenemos el mecanismo necesario para la coordinación de la transmisión del contenido media. De dicho proceso y de la transmisión final de los datos se encarga el componente RTCPeerConection. Dicho componente representa una abstracción del canal de comunicación entre dos clientes. Detallamos el proceso a continuación: Un primer cliente se conecta al servidor de señalización e inicializa el canal. El navegador accede a los dispositivos media de su equipo. Un segundo cliente se conecta al servidor, solicitando entrar al canal creado por el anterior usuario. El navegador accede también al contenido media y envía un mensaje al navegador del primer cliente para iniciar el proceso de negociación. En el navegador del emisor se crea un objeto RTCPeerConection, se añade el stream media capturado y se envía una oferta SDP al receptor a través del servidor de señalización. El proceso en el equipo del receptor es idéntico: se crea un objeto RTCPeerConection, se añade el contenido media capturado y se envía una respuesta SDP al emisor. Durante este proceso el servidor de señalización intercambia información de red (utilizando el protocolo ICE). Una vez el navegador del emisor reciba la respuesta a la oferta, el proceso de negociación ha terminado y es cuando empieza el streaming de contenido media directo entre los dos clientes. El proceso se muestra en la figura 14:

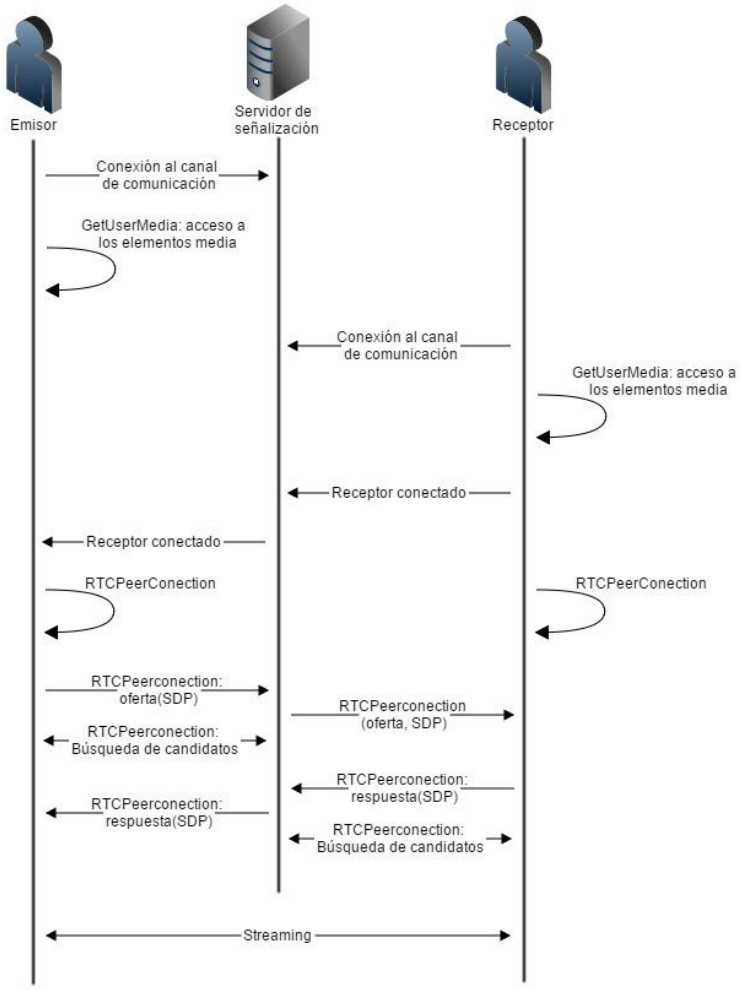


Figura 14: Transmisión de contenido media

Una vez hecho esto, ya tenemos las descripciones locales y remotas configuradas en las conexiones del emisor y del receptor, además de la información de red de ambos clientes, por lo que ya puede empezar el proceso de comunicación peer-to-peer entre ambos clientes.

7.2.3 Streaming de datos

Además de contenido media, por el canal establecido en el paso anterior se puede enviar datos de carácter genérico no media. Esto se consigue haciendo uso del componente RTCDataChannel[20]. El objeto RTCDataChannel representa un canal de comunicación bidireccional directo entre dos clientes y es creado por el cliente que crea el canal como parte del mismo, mediante la función createDataChannel. Esta función acepta dos parámetros: un string que identificará al canal y una variable opcional con parámetros para la configuración del mismo [LR14]. Mediante estas opciones podemos asignar distintos niveles de confiablidad al canal, asegurándonos que los datos se entregan al usuario o estableciendo un número máximo de transmisiones o tiempo de espera.

La transmisión de los datos se hace haciendo uso del protocolo SCTP para el control del flujo y secuenciación de la transmisión de los datos, DTLS para la seguridad y UDP para la transmisión NAT.

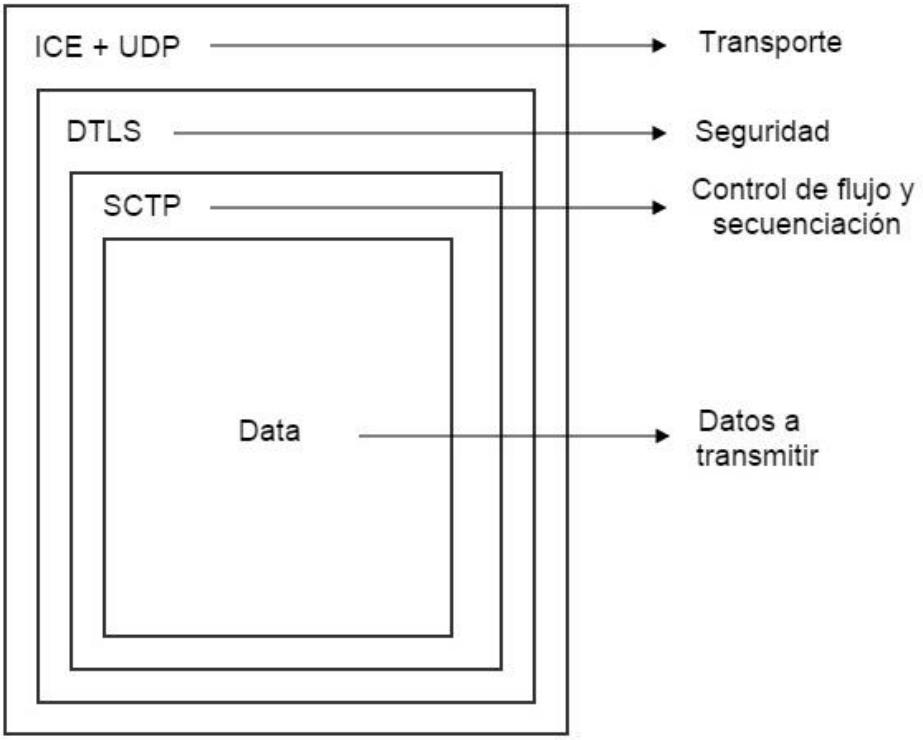


Figura 15: Modelo del streaming de datos

[20] http://w3c.github.io/webrtc-pc/#rtcdatachannel

El protocolo ICE, como hemos visto en apartados anteriores, nos proporciona un mecanismo de descubrimiento de interfaces de red para el envío directo de los datos.

DTLS[21] es un protocolo que nos proporciona privacidad, confidencialidad, integridad y autentificación de fuentes en el envio de datagramas, previniendo la falsificación o visualización de los mensajes.

SCTP[22] es un protocolo de comunicación de la capa de transporte y provee confiabilidad, control de flujo y secuenciación, permitiendo la entrega de mensajes fuera de orden.

La creación de un canal viene asociada a la creación del objeto RTCPeerConection mediante la función createDataChannel. Una vez el emisor ha establecido el canal, notificamos al receptor mediante el servidor de señalización. El evento onDataChannel nos advierte de la creación de dicho canal. En ambos extremos de la comunicación se establecen los manejadores para la apertura de la conexión de datos (onOpen), el cierre (onClose) y la recepción de los mensajes (onMessage). Es entonces cuando puede comenzar la transmisión peer to peer de datos que mostramos en la figura 16.

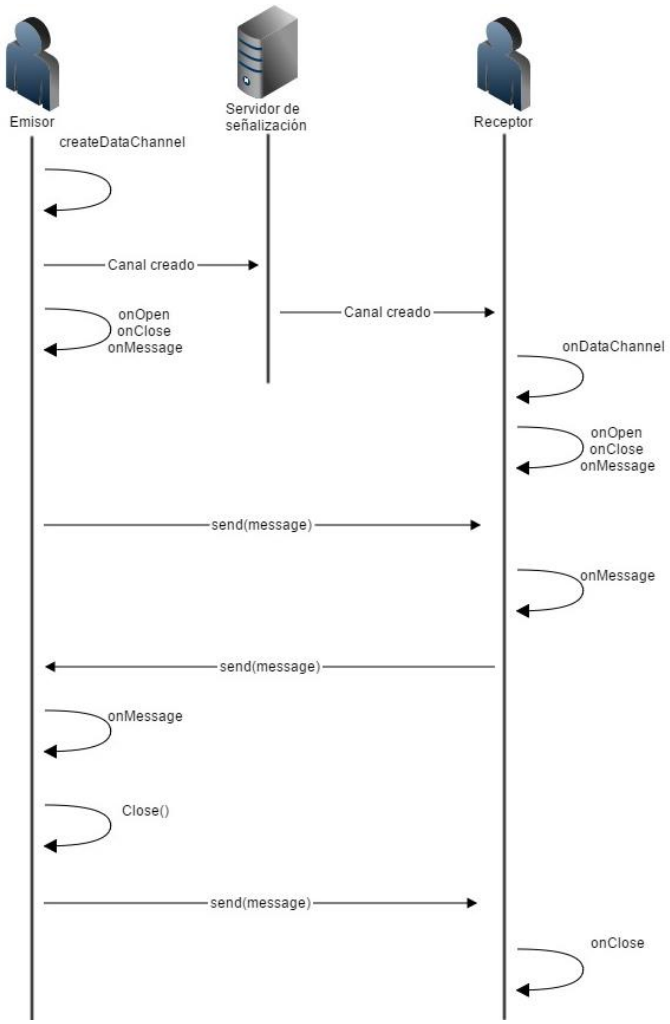


Figura 16: Streaming de datos

[21] https://tools.ietf.org/html/rfc6347
[22] https://tools.ietf.org/html/rfc4960

7.2.4 Exportación e integración

Uno de los requisitos que nos planteamos al comienzo del desarrollo de la librería de comunicación fue la posibilidad de integración de la misma en otros proyectos. El proceso es tan sencillo como crear los elementos HTML5 en el proyecto cliente y cargar las librerías necesarias de nuestro proyecto KairosOVC, tal y como se detalla en el siguiente trozo de código:

```
<!-- Elementos para el video -->
<video id='localVideo' autoplay muted></video>
<video id='remoteVideo' autoplay></video>

<!-- Elementos para el chat -->
<textarea id="dataChannelReceive"></textarea>
<textarea id="dataChannelSend"></textarea>
<button id="sendButton"> Enviar </button>

<!-- Carga de la librería -->
<script src='http://192.168.1.50:2013/socket.io/socket.io.js'> </script>
<script src='http://192.168.1.50:2013/adapter.js'></script>
<script src='http://192.168.1.50:2013/main.js'></script>
```

7.2.5 Actualización del entorno tecnológico

El entorno tecnológico de nuestro proyecto KairosOVC actualizado y que mostramos en la figura 17 queda:

- Socket.io: módulo de Node para el manejo de eventos en tiempo real a través de una conexión TCP.

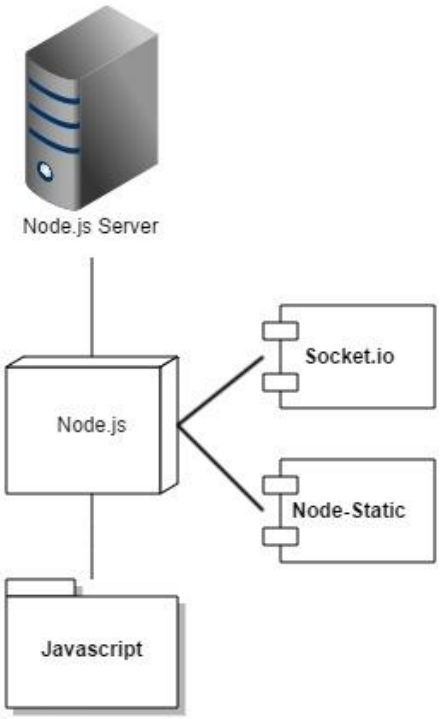


Figura 17: Entorno tecnológico

7.3 Sprint 3: Acceso y consulta del paciente

Una vez definidas las herramientas, el entorno tecnológico e implementado el sistema de comunicación por video-chat, empezaremos el desarrollo de nuestro proyecto cliente KairosOMAC. En esta iteración empezaremos a introducir las tareas definidas por el product owner en el product backlog: seleccionaremos un subconjunto de tareas y compondremos el sprint backlog en forma de historias de usuario. Realizaremos un documento de análisis funcional que será refinado y validado por nuestro product owner. Una vez hayamos gestionado las expectativas del usuario con respecto al análisis y los objetivos del sprint, realizaremos el diseño técnico de la solución y la implementación, obteniendo un prototipo funcional que será validado también por nuestro product owner.

7.3.1 Historias de usuario

Durante esta iteración trabajaremos en el acceso a la aplicación por los diferentes usuarios y en el proceso de consulta del paciente, que consiste en la gestión de sus citas y el acceso a la consulta por video-chat. En este punto mencionar que aunque a la funcionalidad de acceso nuestro product owner no le haya dado una prioridad muy alta, ésta nos hace falta para el desarrollo del resto, por lo que mediante un proceso de negociación acordamos introducirla en este primer sprint.

Identificador	1	Product Backlog ID	15, 16
Título	Acceso a la aplicación		
Descripción:	Como usuario quiero acceder a la aplicación para tener acceso a la funcionalidad que me proporciona.		
Prioridad (1 a 10):	1	Dependencias	

Identificador	2	Product Backlog ID	4
Título	Listado de citas del paciente		
Descripción:	Como paciente quiero ver el listado de mis citas para poder acceder a la consulta.		
Prioridad (1 a 10):	5	Dependencias	1

Identificador	3	Product Backlog ID	1
Título	Video-consulta del paciente		
Descripción:	Como paciente quiero acceder a una sesión de consulta con el médico para poder recibir un diagnóstico y un tratamiento.		
Prioridad (1 a 10):	10	Dependencias	1, 2

7.3.2 Acceso a la aplicación

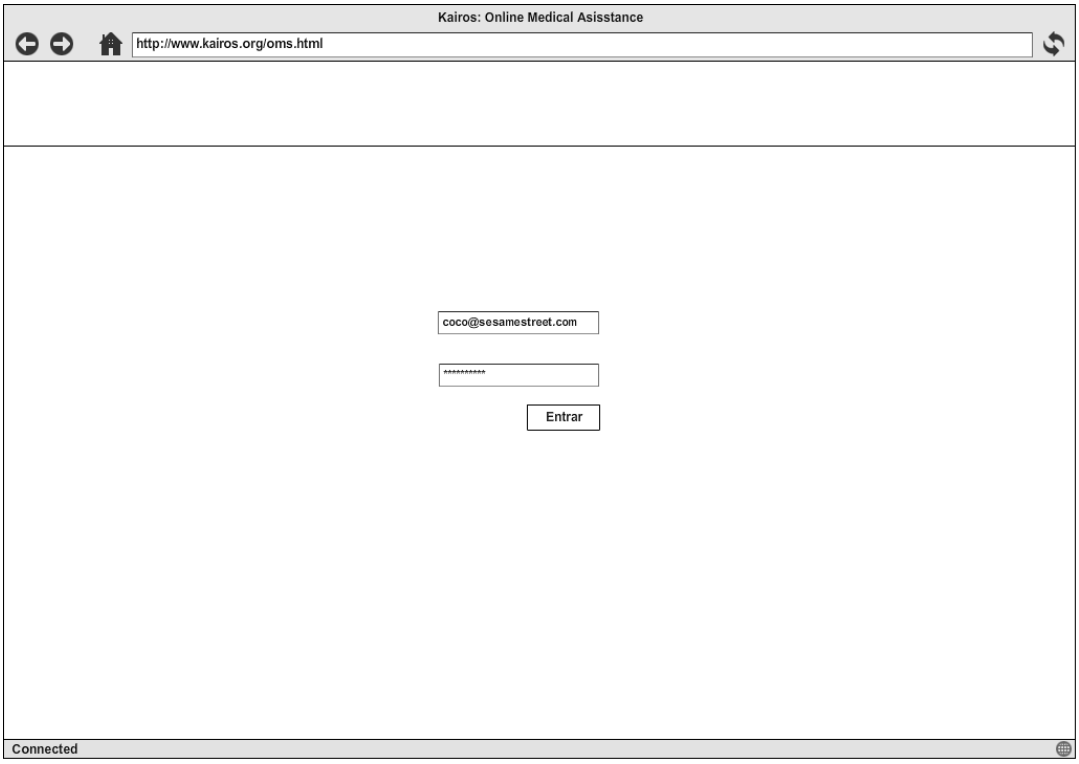
Análisis funcional

Descripción

Para acceder a la aplicación el usuario deberá identificarse mediante una combinación usuario/contraseña. Una vez comprobada su acreditación el usuario será redirigido a la pantalla de gestión de citas si es un usuario de tipo paciente, o a la pantalla de gestión de consultas si es un médico.

Mockups

La pantalla de acceso presenta dos campos de entrada de texto, uno para el nombre de usuario y otro para la contraseña, además de un botón cuya funcionalidad es la de proceder a la comprobación de acreditación del usuario. La pantalla se representa en el mockup 1:



Mockup 1: Acceso a la aplicación

Usuarios

- Paciente.
- Médico.

Historias de usuario

- 1.

Restricciones

- Los usuarios deben estar dados de alta en la aplicación para poder acceder a ella.
- El acceso al sistema se hará en base a la combinación email/password.
- No podrá haber dos usuarios en el sistema con el mismo email.
- El campo contraseña debe ocultar los caracteres que esté escribiendo el usuario.

Pruebas de aceptación

- El paciente se logea correctamente y accede a la pantalla de gestión de citas.
- El doctor se logea correctamente y accede a la pantalla de gestión de consultas.
- El usuario, médico o paciente, se logea incorrectamente, se muestra mensaje de error y nos quedamos en la pantalla de login.

Diseño

Para el acceso a la aplicación haremos uso de un plugin de Codeigniter llamado Ion_auth[23]. Ion_auth es una librería con funcionalidad para la gestión de autentificación y manejo de roles de usuario. Nos proporciona un sistema de autentificación seguro basado en el almacenamiento de los passwords con **¡Error! No se encuentra el origen e la referencia.** y evitando ataques **¡Error! No se encuentra el origen de la referencia.**. Es una mejora de la librería Redux Auth 2 y hoy en día es una de las más utilizadas en el desarrollo de aplicaciones bajo Codeigniter debido a su sencillez de uso, su simplicidad y su gran potencialidad. En esta librería delegaremos la autentificación y gestión de roles de nuestra aplicación. Además Ion_auth proporciona funcionalidad para la gestión avanzada de usuarios, funcionalidad que implementaremos en iteraciones avanzadas de nuestro desarrollo.

La instalación es tan sencilla como descargar el último paquete de la distribución (2.0 en el momento de la realización del proyecto), sobrescribir nuestro código y ejecutar el SQL correspondiente al motor de base de datos que estemos utilizando (MySQL en nuestro caso).

Ion_auth utiliza una organización de usuarios basada en pertenencia a grupos. El diagrama entidad-relación 1, que representa esta asociación, se muestra a continuación:

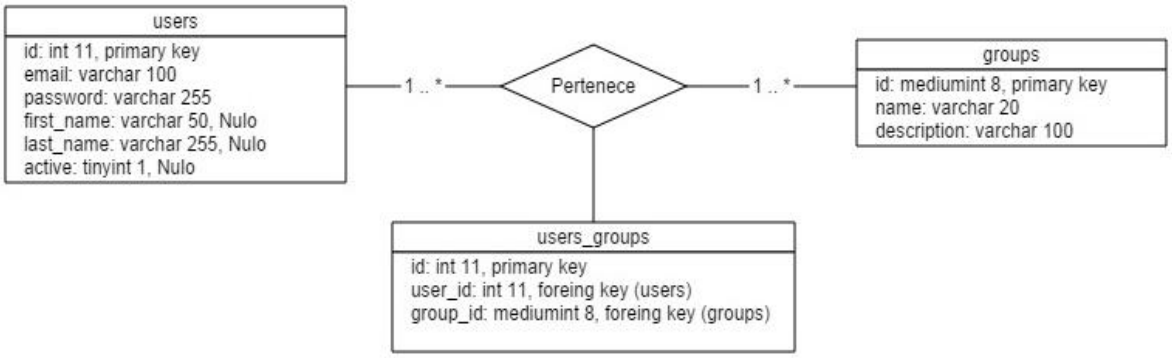


Diagrama entidad-relación 1: Usuarios - grupos

Describiremos ahora las acciones involucradas en el proceso de logeo. El usuario de la aplicación, médico o paciente, accede a la web y procede a identificarse. Comprobamos entonces si el usuario está dado de alta en el sistema para que, en caso afirmativo, sea redirigido a la pantalla de gestión principal del tipo al que pertenece. El diagrama de secuencia 1, que representa el proceso de acceso a la aplicación es el siguiente:

[23] http://benedmunds.com/ion_auth/

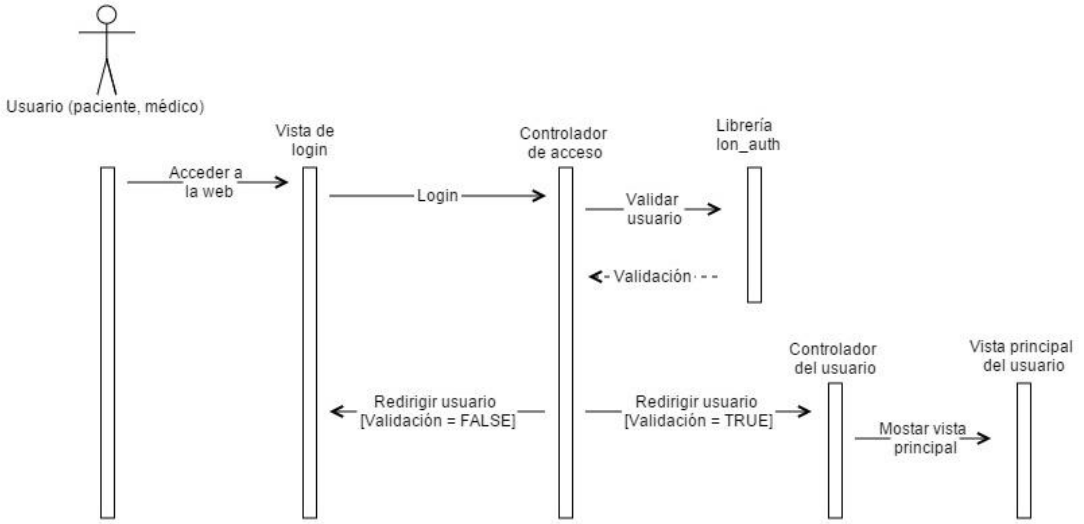


Diagrama de secuencia 1: Acceso a la aplicación

Modelamos ahora el diagrama de secuencia para obtener el modelo clases y tener una visión concreta y cercana al detalle de la implementación de los objetos involucrados en el proceso. Desde la vista de acceso a la aplicación *vlogin*, llamamos al controlador que se encargará de la gestión de usuarios, *auth*. Mediante la función *login* y haciendo uso de la librería de autorización, el controlador comprobará si los datos pasados pertenecen a un usuario válido para redirigirlo al método principal, *index*, del controlador del tipo al que pertenece, *cpatients* o *cdoctors* y redirigirlos a la vista de gestión apropiada: *vpatient* o *vdoctor* respectivamente. El diagrama de clases 1 representando el modelo explicado:

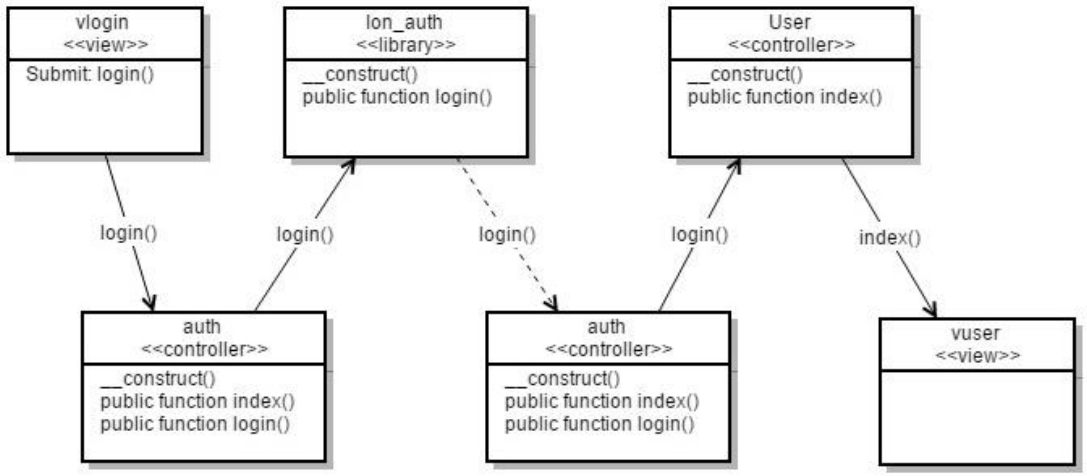
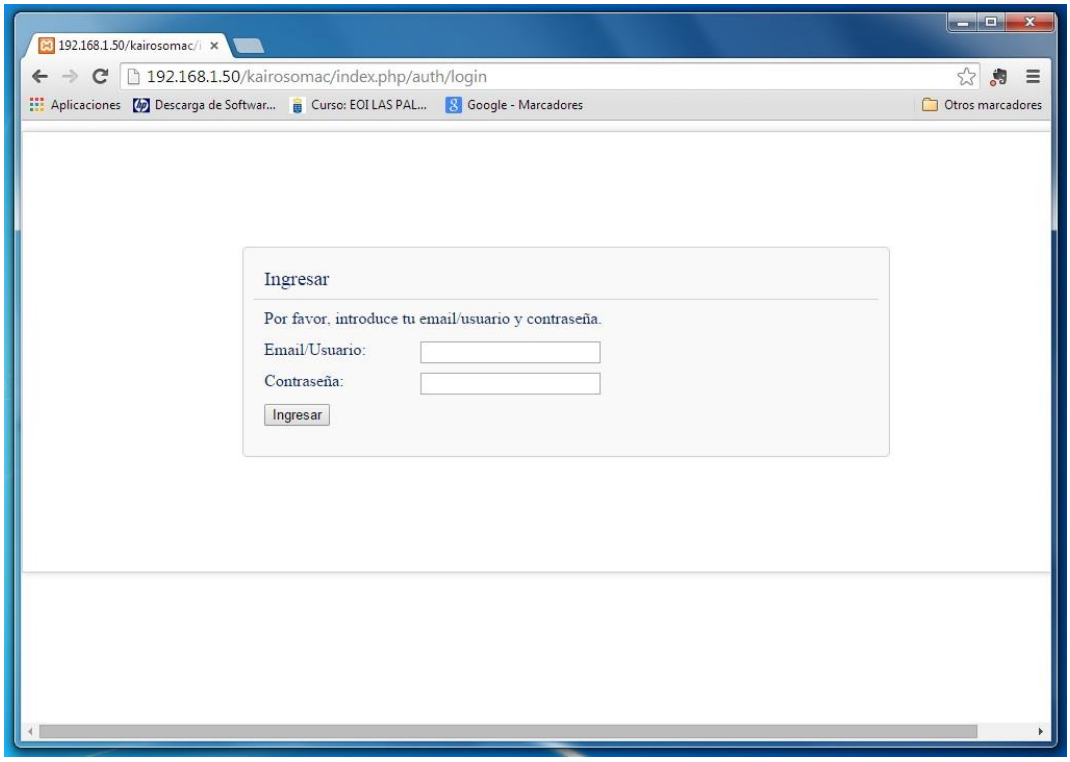


Diagrama de clases 1: Acceso a la aplicación

Pantallas

La pantalla 1, de acceso a la aplicación.



Pantalla 1: Acceso a la aplicación

7.3.3 Listado de citas del paciente

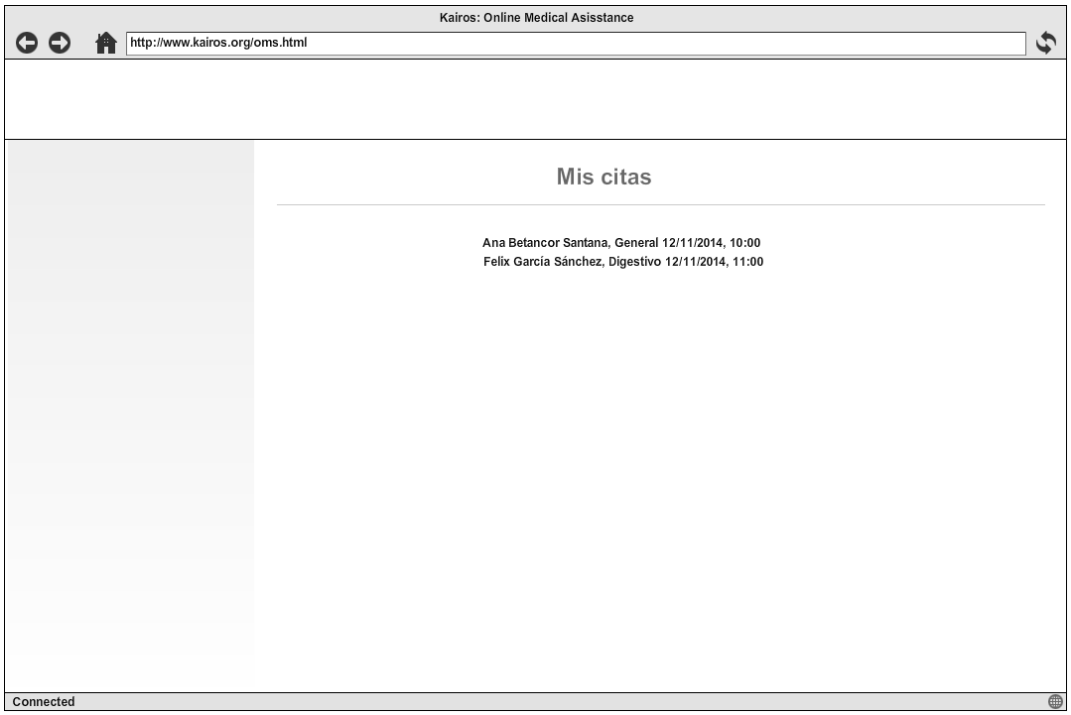
Análisis funcional

Descripción

Se muestra un listado con las citas para día actual del paciente. La información que se mostrará en las citas será la del nombre del médico, su especialidad, la fecha y hora de la consulta.

Mockup

La pantalla principal se dividirá en dos. En la parte derecha dejaremos un espacio para un futuro menú, en la parte central mostraremos la información de la pantalla en la que estemos, que en este caso será el listado de citas para el día actual. El mockup 2:



Mockup 2: Listado de citas del paciente

Usuarios

- Paciente.

Historias de usuario

- 2.

Restricciones

- Sólo se muestran las citas del paciente actual.
- Sólo se muestran las citas del día en cuestión.

Pruebas de aceptación

- Al acceder se muestran todas las citas del usuario para el día actual.

Diseño

La idea es implementar la pantalla principal del menú del paciente. La pantalla se mostrará cuando el paciente se haya logeado con éxito y mostrará el listado de citas para el día actual. Lo primero es establecer los modelos de los objetos implicados en el proceso: paciente, doctor y cita. El modelo del objeto paciente, que será una especialización del objeto usuario, se presenta a continuación en el diagrama entidad-relación 2:

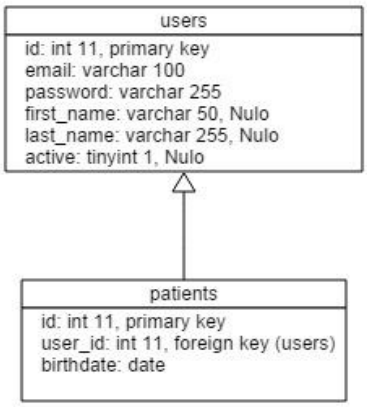


Diagrama entidad-relación 2: Usuario-paciente

Por el momento el único campo propio es el de fecha de nacimiento. El modelo del objeto doctor, que también es una especialización del objeto usuario, tiene como elemento diferenciador la especialidad del mismo. La representación en el diagrama 3:

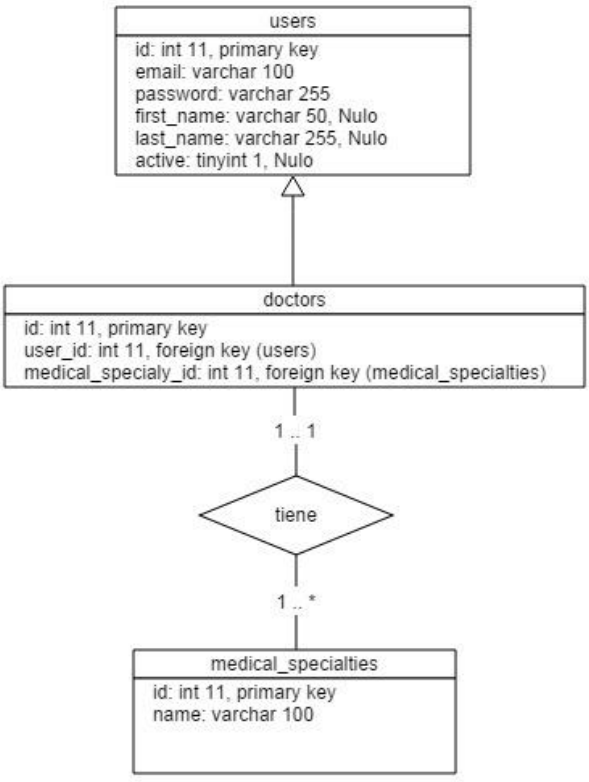


Diagrama entidad-relación 3: Usuario-doctor

El objeto cita, que representa un arreglo en un instante de tiempo entre el paciente y el doctor con la finalidad de estudiar unos síntomas y emitir un diagnóstico se representa de la siguiente forma en el diagrama 4:

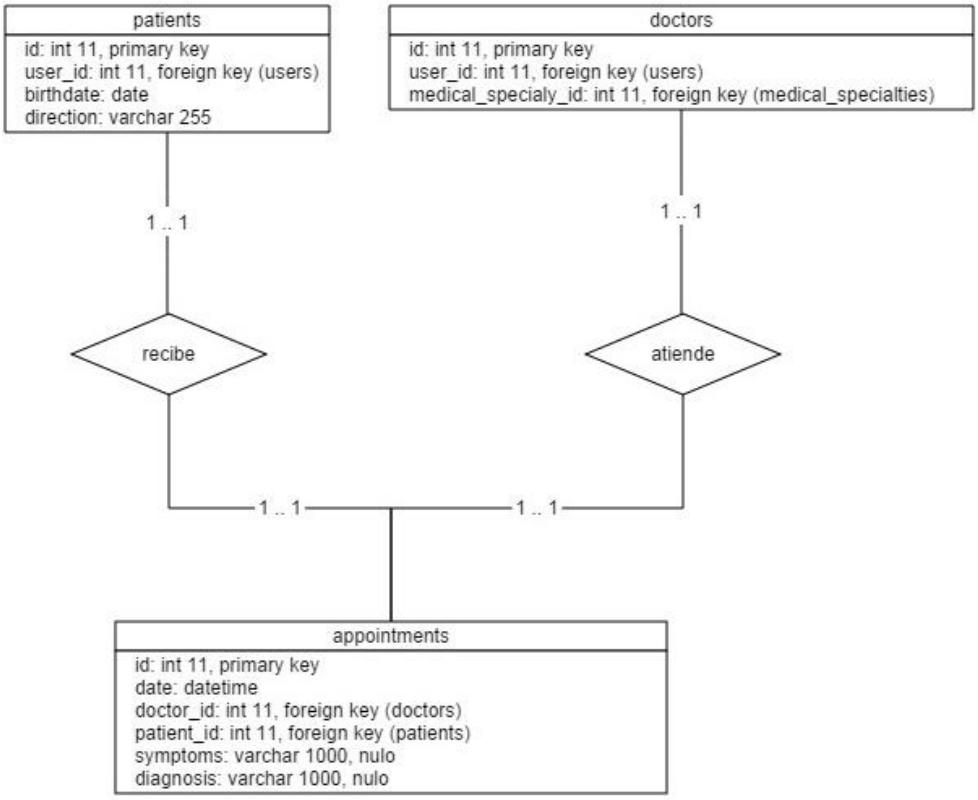


Diagrama entidad-relación 4: Paciente - doctor - cita

Una vez construidos los modelos pasamos a definir el proceso. El paciente, después de logearse correctamente, es redirigido a la vista de gestión con la información de las citas, proceso mostrado en el diagrama de secuencia 2:

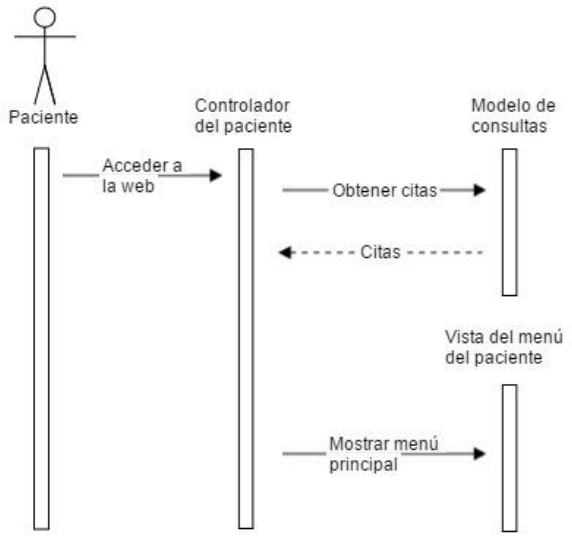


Diagrama de secuencia 2: Obtener citas del paciente

Extrapolemos ahora el diagrama de secuencias al diagrama de clases 2, con los métodos necesarios para obtener y mostrar el listado de citas. Creamos el controlador *cpatients* para agrupar la implementación de la funcionalidad del modelado del paciente y un paquete para agrupar las vistas del mismo con la vista principal *vpatient*. Además necesitaremos obtener la información relacionada con las citas por lo que también creamos un modelo para las mismas, *appointments_model*. Como vamos a construir la pantalla de gestión principal, lo vamos a hacer desde el método *index* del controlador. Al acceder, cargaremos desde el modelo las citas con el método *get_appointments_by_patient*

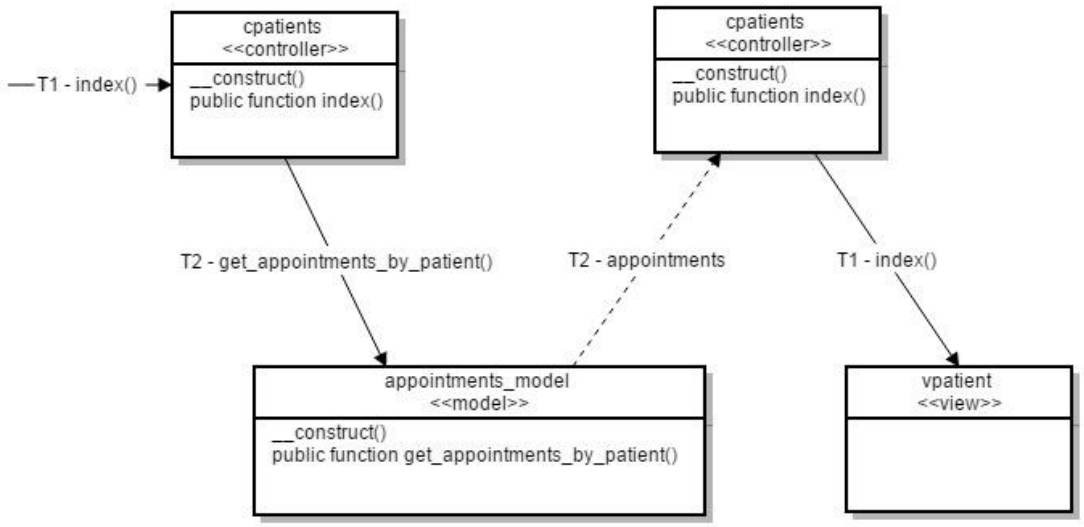
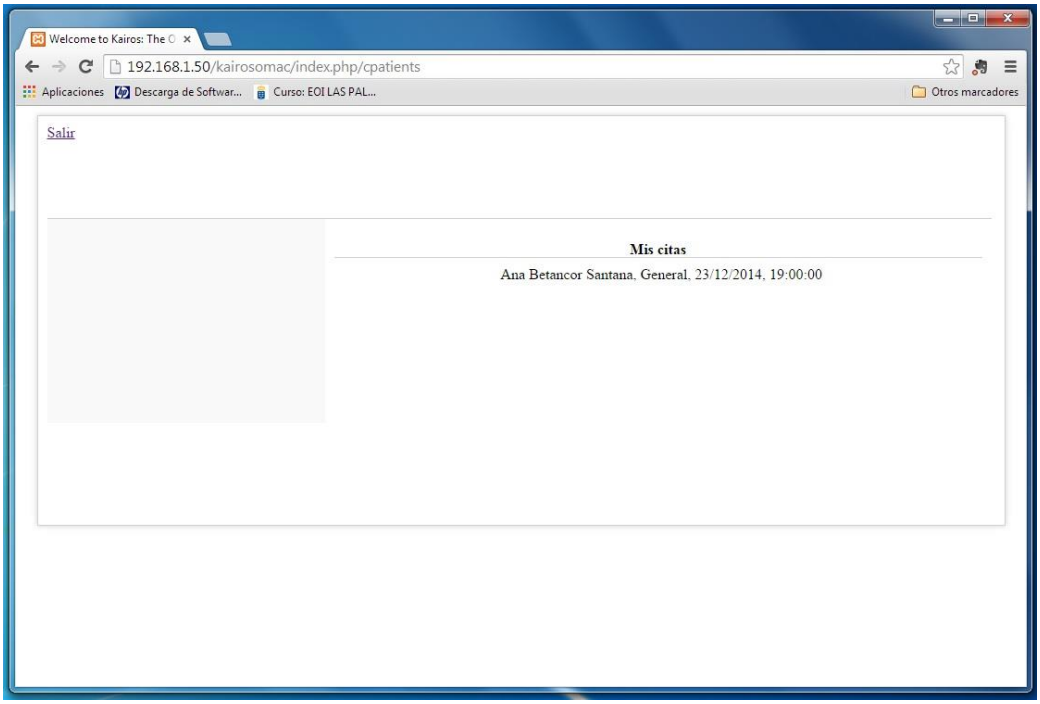


Diagrama de clases 2: Obtener citas del paciente

Pantallas

La pantalla 2 finalizada del paciente, mostrando las citas del día.



Pantalla 2: Listado de citas del paciente

7.3.4 Consulta del paciente

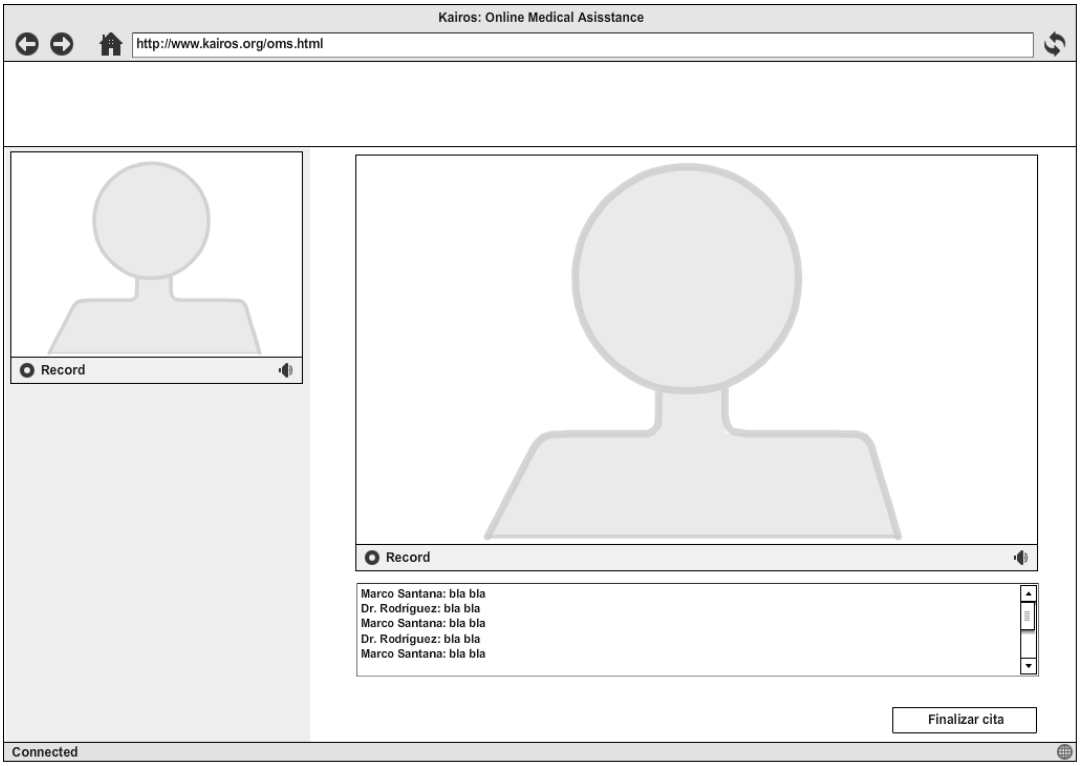
Análisis funcional

Descripción

Cuando el paciente pulse sobre una de la citas de la pantalla anterior será redirigido a una vista nueva para que pueda tener una consulta con el médico mediante video-chat. El paciente puede terminar la cita cuando quiera, lo que le llevaría de regreso a la pantalla de gestión de citas.

Mockup

Dividiremos la pantalla en dos secciones. En la principal mostraremos la captura del video del doctor, las opciones de chat y un botón para finalizar la consulta. En la sección de la derecha la captura del video del paciente. La estructura representada en el mockup 3:



Mockup 3: Consulta del paciente

Usuarios

- Paciente.

Historias de usuario

- 3.

Restricciones

- La consulta entre médico y paciente debe ser exclusiva.

Pruebas de aceptación

- El paciente accede a la nueva pantalla de consulta pulsando sobre uno de los enlaces de su listado de citas.
- Se muestra el video del paciente.
- Se vuelve a la pantalla principal del paciente al pulsar sobre el botón para finalizar la consulta.

Diseño

Debemos tener en cuenta que la sesión entre paciente y doctor debe ser exclusiva por lo que tenemos que idear un mecanismo para identificarla. La opción elegida es una combinación entre los identificadores de paciente y doctor.

La secuencia de acciones correspondientes y representadas en el diagrama de secuencia 3 sería la siguiente: el paciente desde su vista principal pulsaría sobre la sesión de consulta a la que quiere acceder, lo que lo llevaría a la vista de consulta por video-chat.

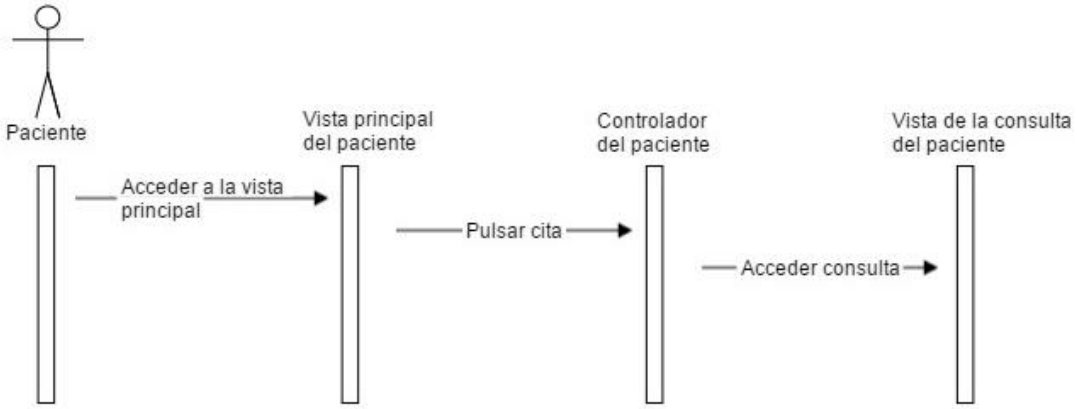


Diagrama de secuencia 3: Acceder a la consulta

Debemos mostrar las citas listadas en la vista *vpatient* como links que enlacen a un método del controlador que llamaremos *appointment*, que redirija a la nueva vista de consulta *vappointment*, en la que integraremos la solución implementada para realizar una comunicación por video-chat. El proceso modelado en el diagrama de clases 3:

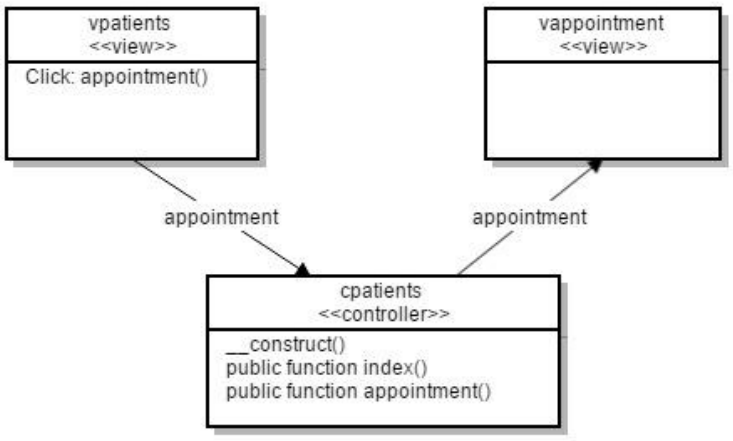
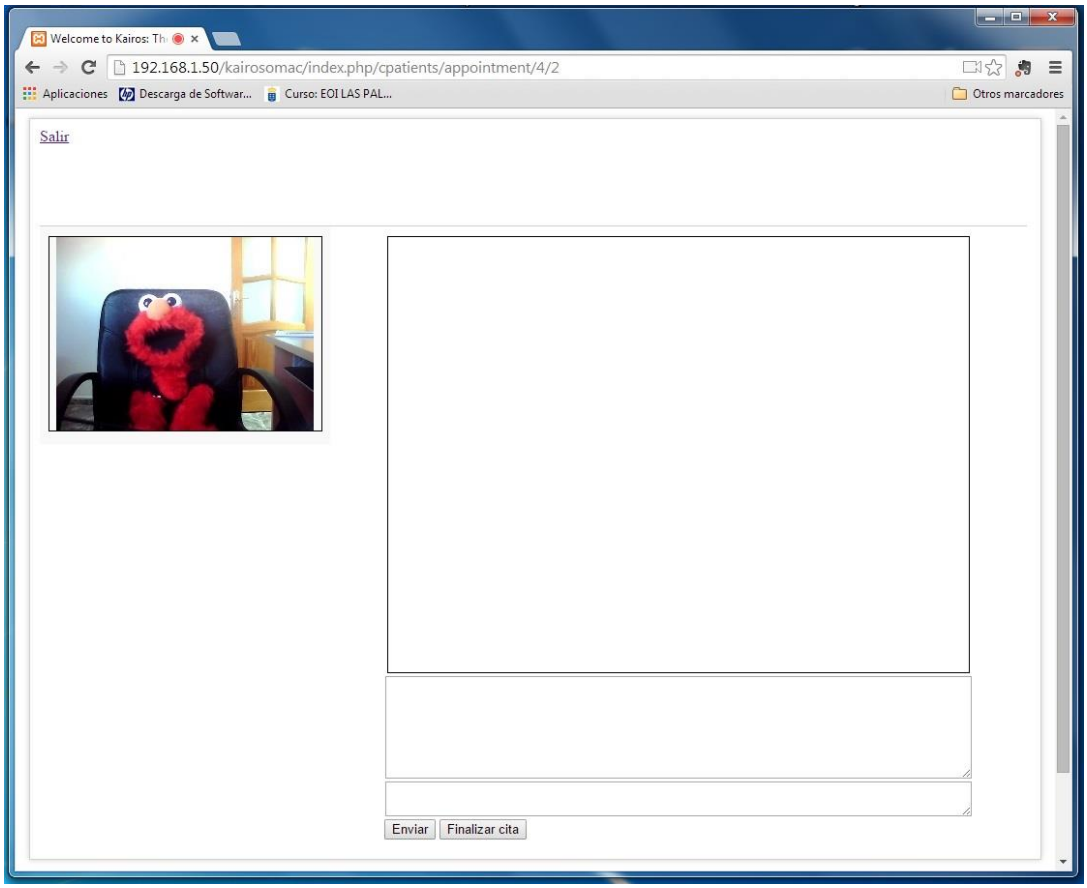


Diagrama de clases 3: Acceder a la consulta

Pantallas

La pantalla 3, de consulta del paciente, con el video local capturado a la izquierda, el chat y la vista donde irá el video capturado del doctor.



Pantalla 3: Consulta del paciente

7.3.5 Actualización del entorno tecnológico

A continuación mostramos en la figura 18, el entorno tecnológico con la adición de los nuevos componentes que hemos utilizado en esta iteración:

- Ion_auth: librería de Codeginiter para el manejo de la gestión de usuarios.

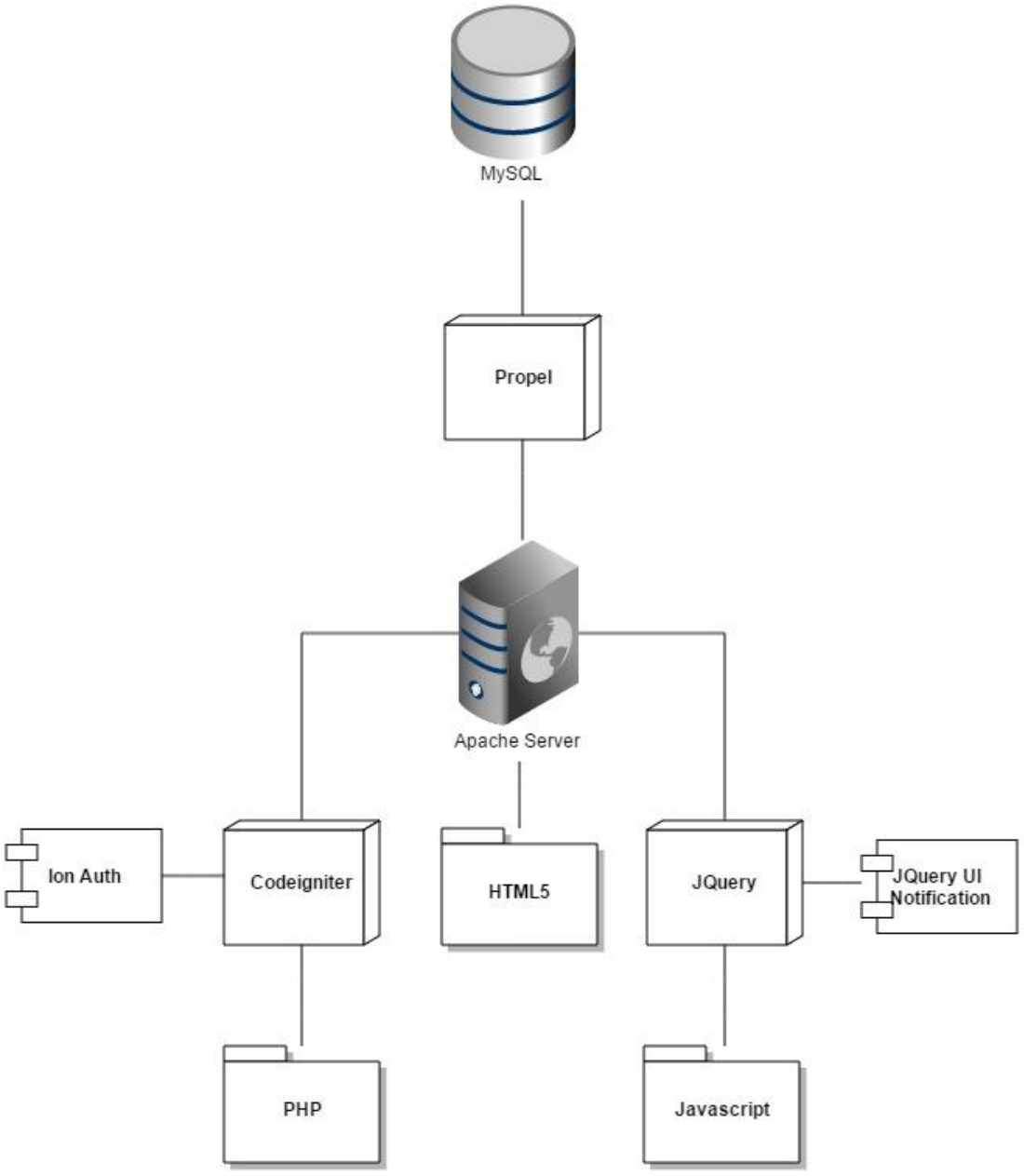


Figura 18: Entorno tecnológico

7.4 Sprint 4: Consulta del médico

En este sprint abordaremos las tareas de gestión de citas y consulta por video-llamada del lado del médico.

7.4.1 Historias de usuario

Identificador	4	Product Backlog ID	4
Título	Gestión de citas		
Descripción:	Como médico quiero ver el listado de citas diarias para poder atender a los pacientes.		
Prioridad (1 a 10):	5	Dependencias	1

Identificador	5	Product Backlog ID	1
Título	Consulta por parte del médico		
Descripción:	Como médico quiero atender una sesión de consulta para poder dar respuesta a un paciente.		
Prioridad (1 a 10):	10	Dependencias	1, 4

7.4.2 Gestión de citas

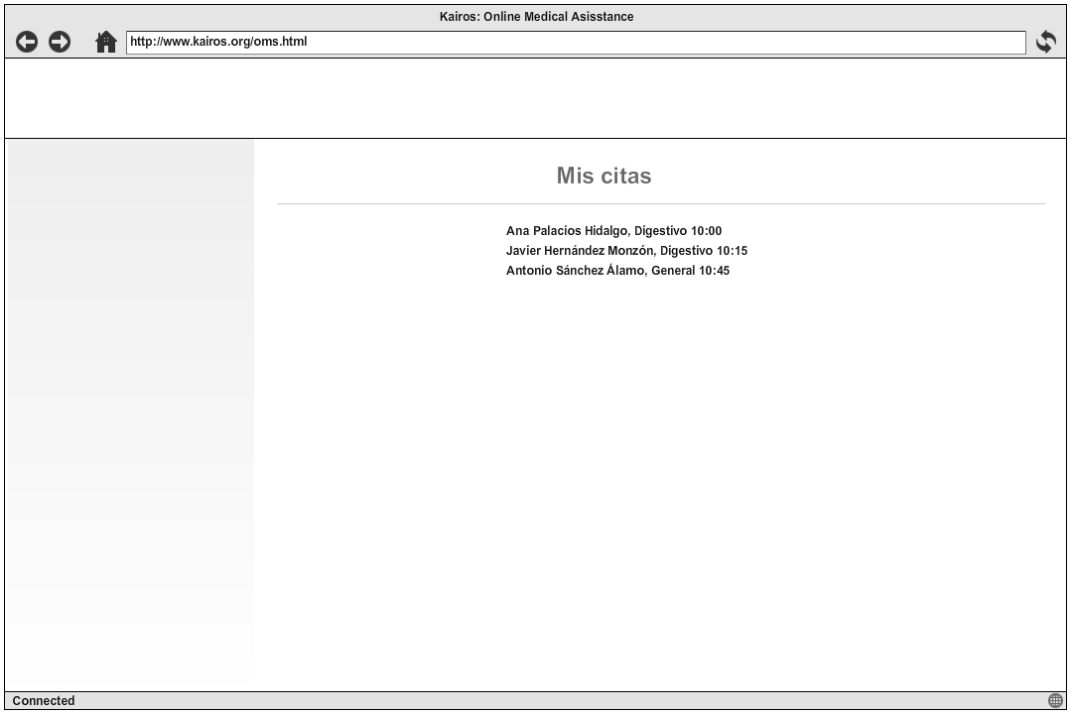
Análisis funcional

Descripción

Desde esta pantalla el médico tiene información de las consultas del día. La información que deberán presentar las citas es la de nombre y apellidos del paciente, especialidad y la hora a la que está programada la consulta.

Mockup

Esta será la pantalla principal del doctor. Se mostrará el listado de citas del día en el centro. El modelo de la pantalla representado en el mockup 4:



Mockup 4: Gestión de citas

Actores

- Médico.

Historias de usuario

- 4.

Restricciones

- Sólo se muestran las citas del día actual.
- Sólo se muestran las citas del médico actual.

Pruebas de aceptación

- Al acceder se muestran todas las citas programadas para el médico para el día actual.

Diseño

El proceso es similar al desarrollado para el paciente. Los objetos que representan al doctor y a la cita médica, así como el modelo de ésta última se definieron en apartados anteriores. El diagrama de secuencia 4 que representa las acciones:

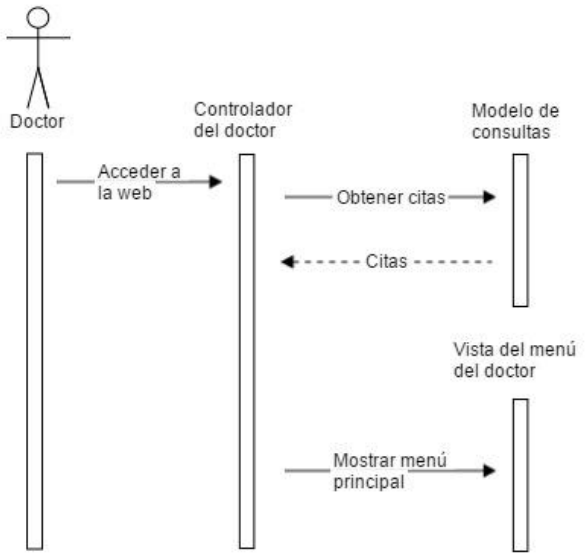


Diagrama de secuencia 4: Obtener citas del médico

El diagrama de clases 4, es también muy similar al de la funcionalidad para los pacientes. Crearemos ahora un controlador *cdoctors*, para la funcionalidad del doctor y una vista para la pantalla de gestión *vdoctor*. El modelo para obtener las citas se creó en apartados anteriores.

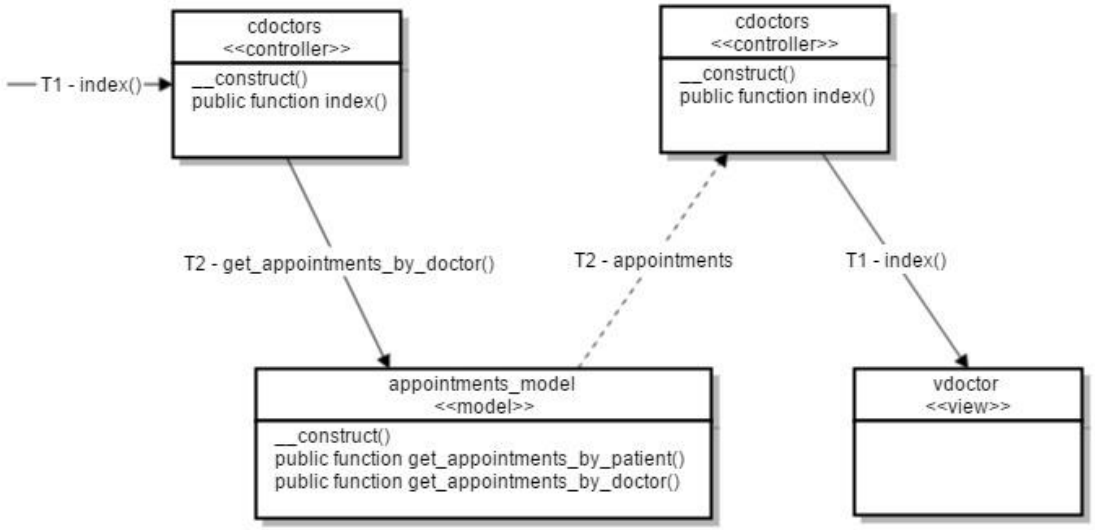
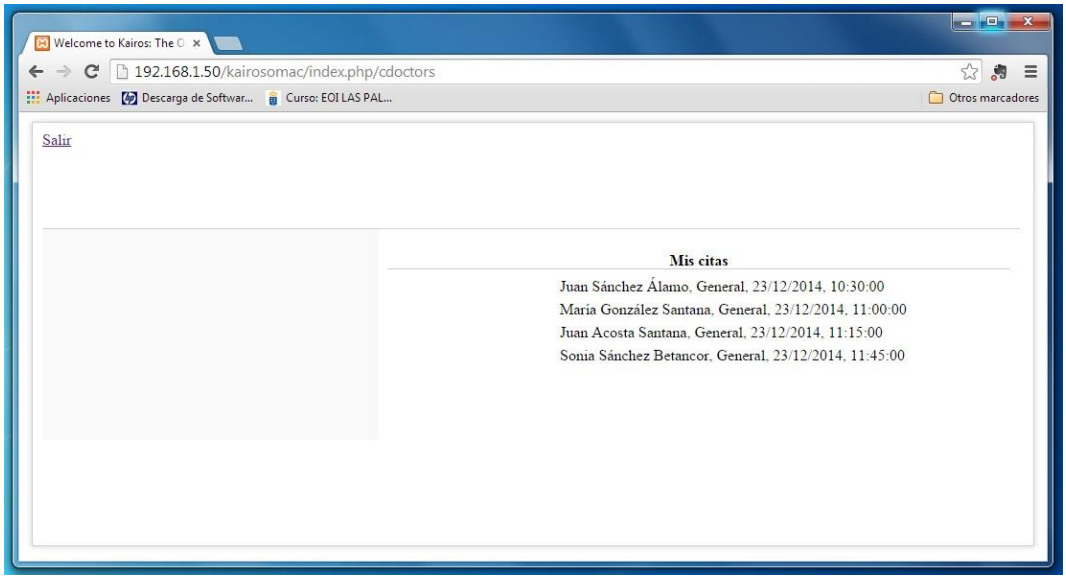


Diagrama de clases 4: Obtener citas del médico

Pantallas

El listado de citas del doctor mostrado en la pantalla 4.



Pantalla 4: Gestión de citas del doctor

7.4.3 Consulta

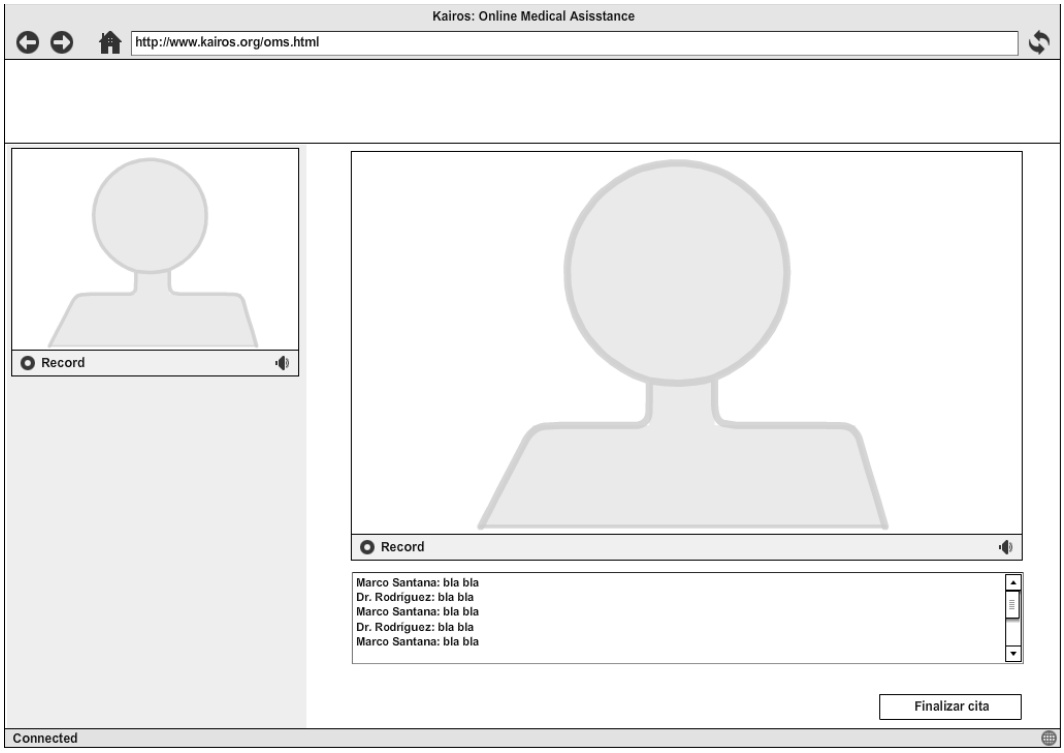
Análisis funcional

Descripción

Cuando el médico pulsa sobre una de la citas de la pantalla anterior, es redirigido a una vista nueva para que pueda tener una consulta con el paciente mediante video-chat. El médico puede terminar la cita cuando quiera, lo que le llevaría de regreso a la pantalla de "Gestión de citas".

Mockup

Dividiremos la pantalla en dos secciones. En la principal mostraremos la captura del video del paciente, las opciones de chat y un botón para finalizar la consulta. En la sección de la derecha la captura de video del doctor. La estructura en el mockup 5:



Mockup 5: Consulta del doctor

Actores

- Médico.

Historias de usuario

- 5.

Restricciones

- La consulta entre médico y paciente debe ser exclusiva.

Pruebas de aceptación

- Se puede realizar una sesión de video-chat con el paciente.
- Se vuelve a la pantalla principal del doctor al pulsar sobre el botón para finalizar la consulta.

Diseño

También para esta pantalla, el comportamiento es similar al del proceso para el paciente, por lo que no entraremos mucho en detalle. El diagrama de secuencia 5:

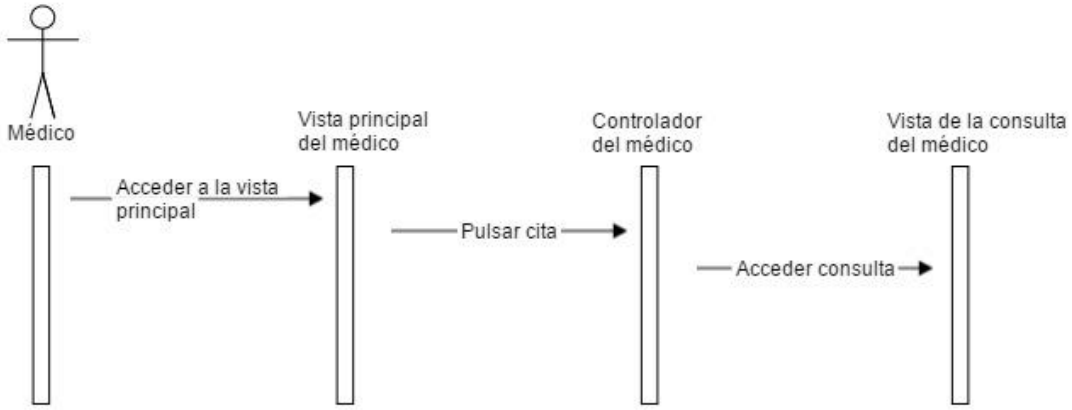


Diagrama de secuencia 5: Acceder a la consulta

Y el diagrama de clases 5, también similar al diseñado para el paciente:

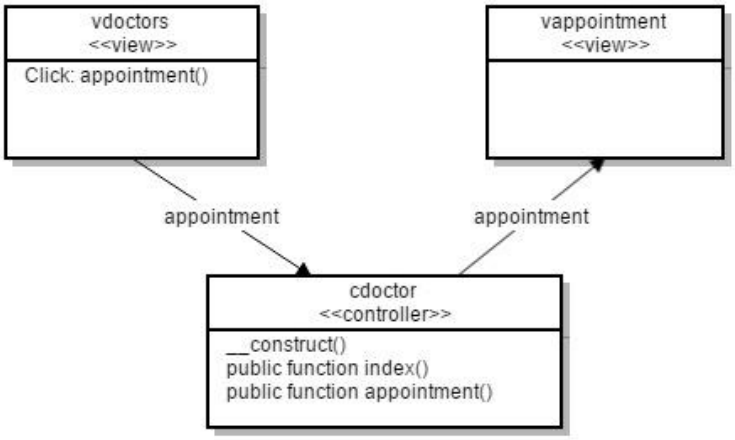
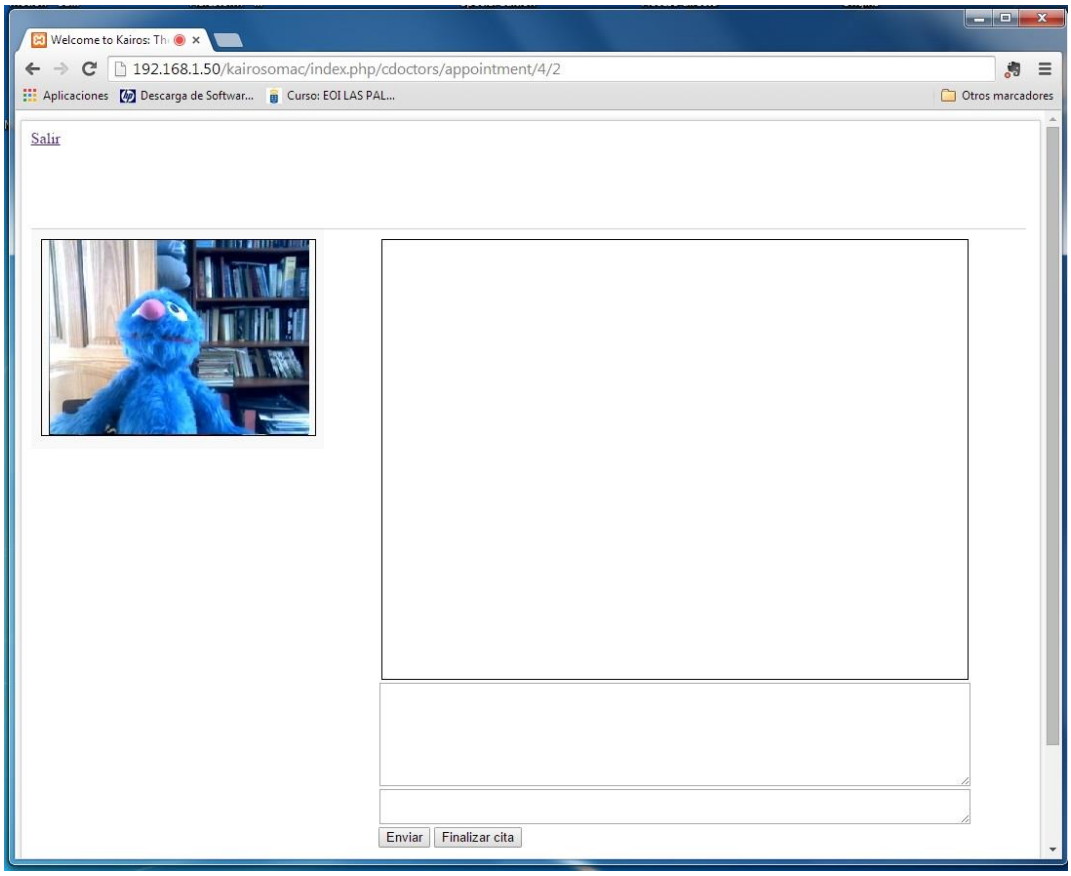


Diagrama de clases 5: Acceder a la consulta

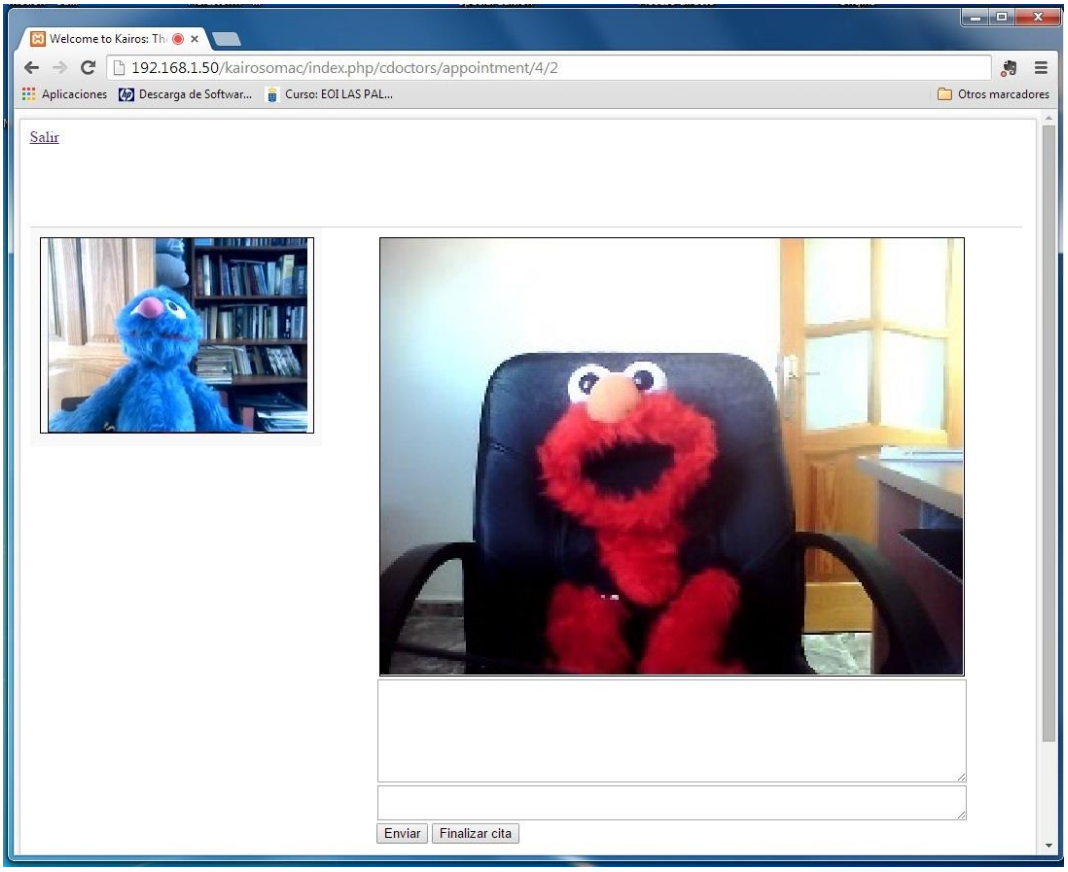
Pantallas

La pantalla 5 de consulta del médico con la captura de video local, las opciones de chat y la vista donde irá el video remoto del paciente:



Pantalla 5: Consulta del doctor

La pantalla 6 muestra la interacción médico-paciente desde el punto de vista del paciente:



Pantalla 6: Interacción médico-paciente

7.5 Sprint 5: Sala de espera y datos de consulta

En este sprint introduciremos dos evolutivos a la funcionalidad de la consulta por video chat: una primera versión de la sala de espera, en la que los pacientes introducen sus citas en una pila y es el doctor el que elige la consulta a efectuar, y una primera versión también de la funcionalidad para guardar los datos del proceso de consulta.

7.5.1 Historias de usuario

Identificador	6	Product Backlog ID	4
Título	Sala de espera		
Descripción:	Como doctor quiero ver que pacientes están en la sala de espera para gestionar el acceso a las consultas.		
Prioridad (1 a 10)	8	Dependencias	3, 5

Identificador	7	Product Backlog ID	3
Título	Detalles de consulta		
Descripción:	Como doctor quiero guardar los datos de una consulta para poder seguir el historial de un paciente.		
Prioridad (1 a 10)	8	Dependencias	3, 5

7.5.2 Sala de espera

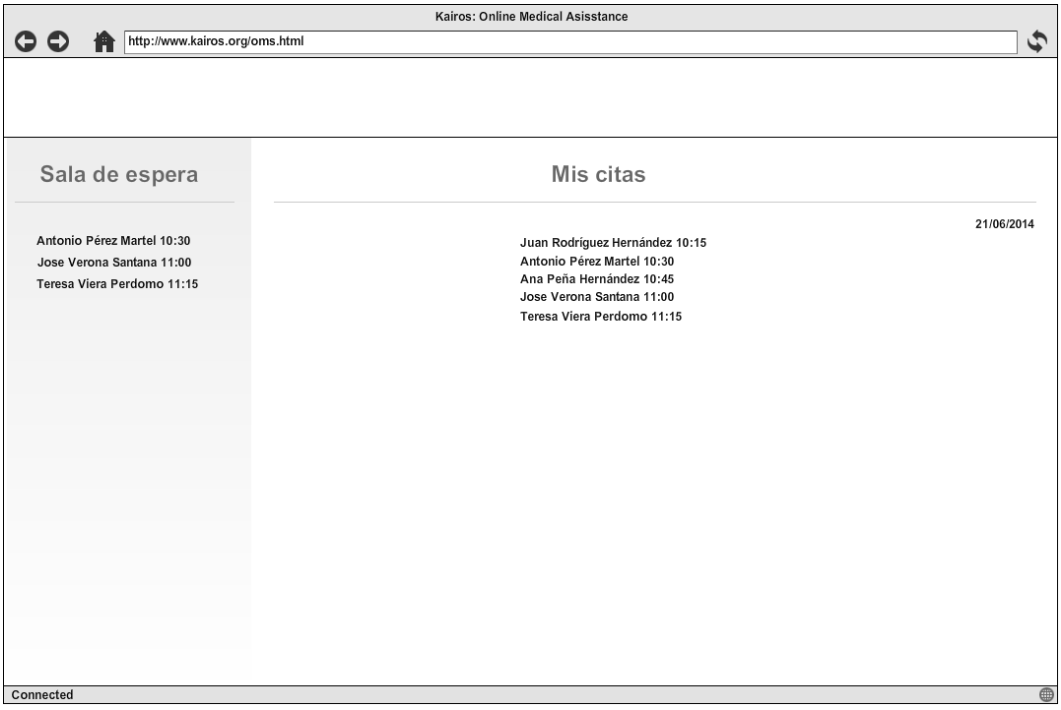
Análisis funcional

Descripción

Cuando un paciente quiere acceder a una sesión de consulta pulsa sobre el link de la cita y ésta se añade a la sala de espera. La pantalla principal del doctor se actualiza con la siguiente información de la cita: nombre del paciente y hora de la consulta. El paciente es redirigido a la pantalla de video-chat y espera a que el médico acceda a la consulta. Para ello, el médico debe pulsar sobre dicha cita de entre las que están que están en la sala de espera.

Mockup

Tenemos una nueva sección en el lateral izquierdo del menú del doctor, con la información de las citas que están en la sala de espera. La sección se irá actualizando a medida que los pacientes vayan pulsando sobre los enlaces de las citas. El diseño de la página se muestra en el mockup 6:



Mockup 6: Sala de espera

Actores

- Paciente
- Médico

Historias de usuario

- 6

Restricciones

- Las citas de la sala de espera sólo podrán ser del día actual.
- Las citas de la sala de espera sólo podrán ser del médico actual.
- La lista de citas se muestra ordenada en base a la hora de las citas.

Pruebas de aceptación

- El paciente pulsa sobre una de las citas, accede a la pantalla de consulta y se actualiza la vista del médico con la nueva cita en la sala de espera.
- El médico pulsa sobre uno de los enlaces de la sala de espera y es redirigido a la pantalla de consulta.

Diseño

La idea es crear un contenedor de citas a la espera de pasar a consulta y que el doctor las gestione. Lo primero es añadir a la cita información del estado en el que se puede encontrar. Estos estados son:

- Nuevo: Cuando se crea una cita.
- En cola: Cuando está en la sala de espera.
- Finalizado: Cuando se ha terminado la consulta.

La representación en el diagrama entidad-relación 5 del estado de una cita:

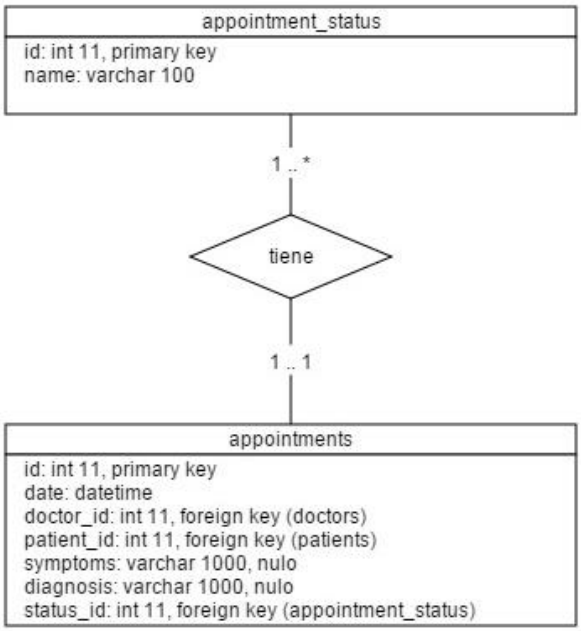


Diagrama entidad–relación 5: Estado cita

Ahora modelamos la sala de espera en el diagrama 6. En ésta, además de guardar la relación con la cita, debemos también guardar un identificador del proceso de comunicación entre paciente y médico, este identificador representa el canal exclusivo por el que se realiza el proceso de video-chat:

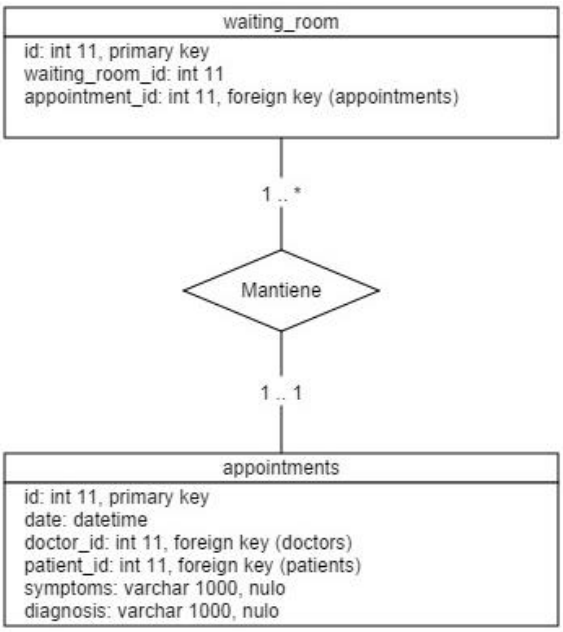


Diagrama entidad-relación 6: Sala de espera

El proceso para añadir una cita a la sala de espera es transparente para el paciente; éste pulsará sobre el link de la cita y será redirigido a la pantalla para la consulta, como venía haciendo hasta ahora. Simultáneamente a esta acción se envía una notificación al doctor indicando que hay una nueva cita. Una vez guardada la cita en la base de datos, la vista del doctor se actualizará con las citas que hay en la sala de espera. El diagrama de secuencia 6 ilustra la secuencia de acciones

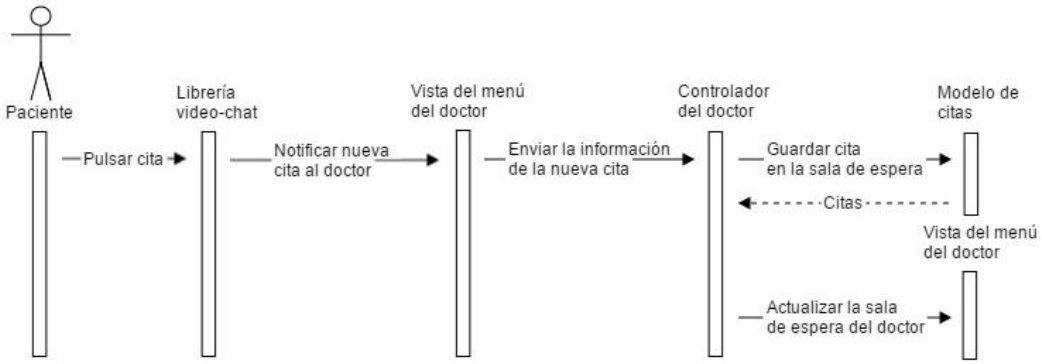


Diagrama de secuencia 6: Añadir cita a la sala de espera

Necesitamos enviar la cita al controlador del doctor cuando el paciente pulse sobre el link de una consulta, por lo que añadimos una nueva función a la vista del paciente, *add_appointment* que enviará la información haciendo uso de nuestro servidor Node. Esta información debe actualizarse de forma asíncrona en la vista del doctor de forma que haremos uso de Ajax para guardar la consulta en base de datos y recargar la vista del doctor, donde se mostraran las citas que están en la sala de espera. Nos crearemos una nueva vista *vappointmets_list* dentro de la vista *vdoctor*, para recargar sólo el listado de pacientes y no la vista entera. La vista del doctor capturara el evento generado

por el servidor con la función *get_appointment* y llamará a la función del controlador para guardar la cita en la base de datos. Una vez guardada, se recuperarán las citas del doctor con *get_appointment_from_waiting_room* y se recargará el listado de citas en la vista del doctor. Por último, tanto para el paciente como para el doctor implementaremos la función *end_appointment* que eliminará una cita de la sala de espera y modificará su estado a finalizada. El diagrama de clases 6, que representa el modelo:

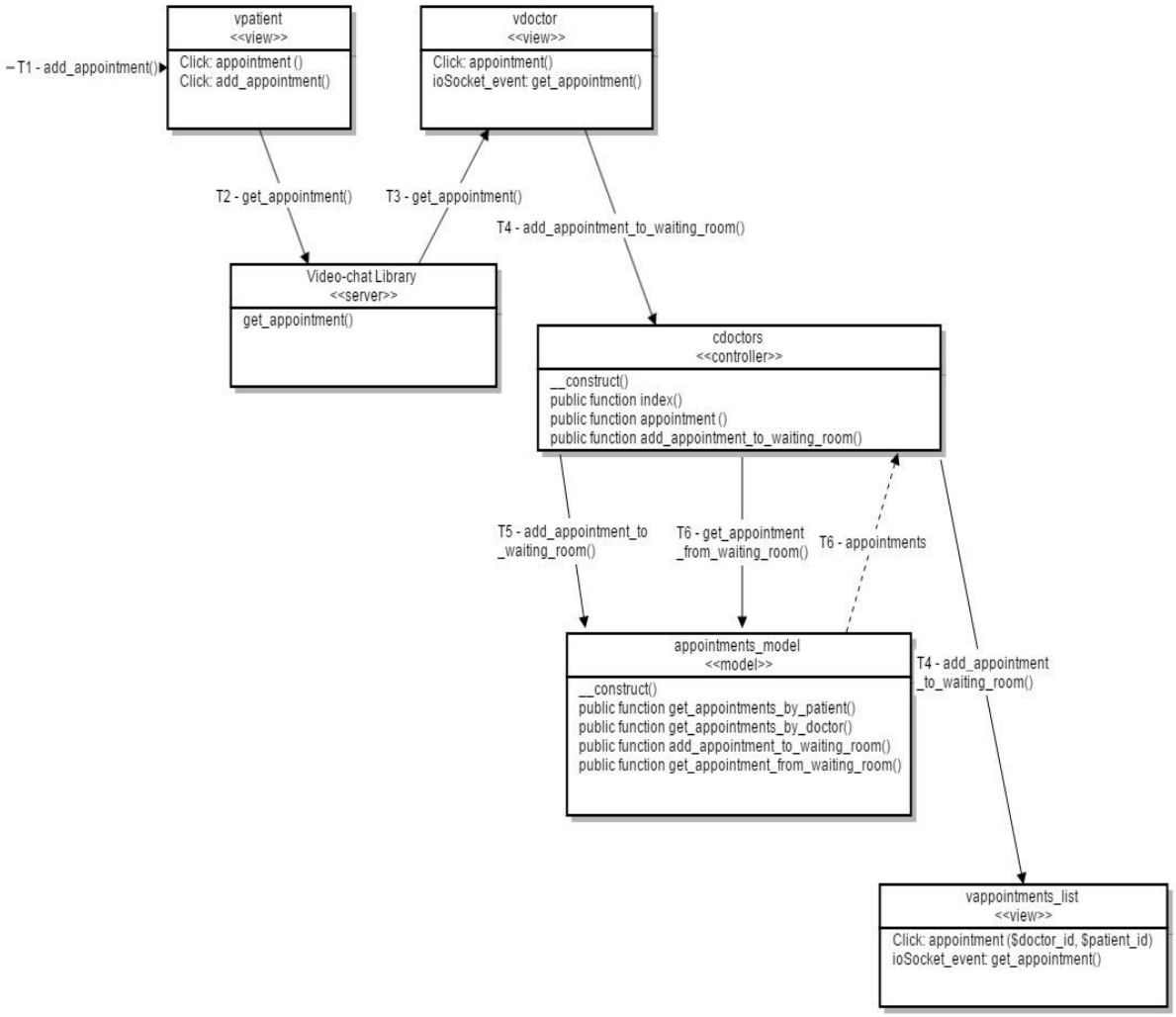
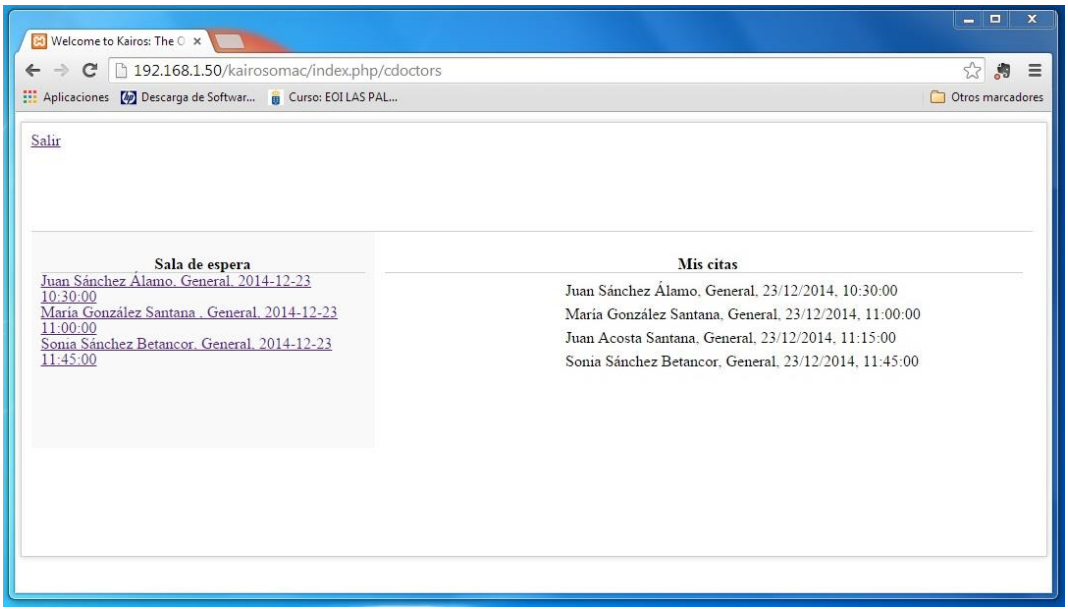


Diagrama de clases 6: Sala de espera

Pantallas

La pantalla 7 del doctor mostrando las citas del día en la zona central y las citas que están en la sala de espera esperando por la consulta en el lateral



Pantalla 7: Sala de espera

7.5.3 Detalles de consulta

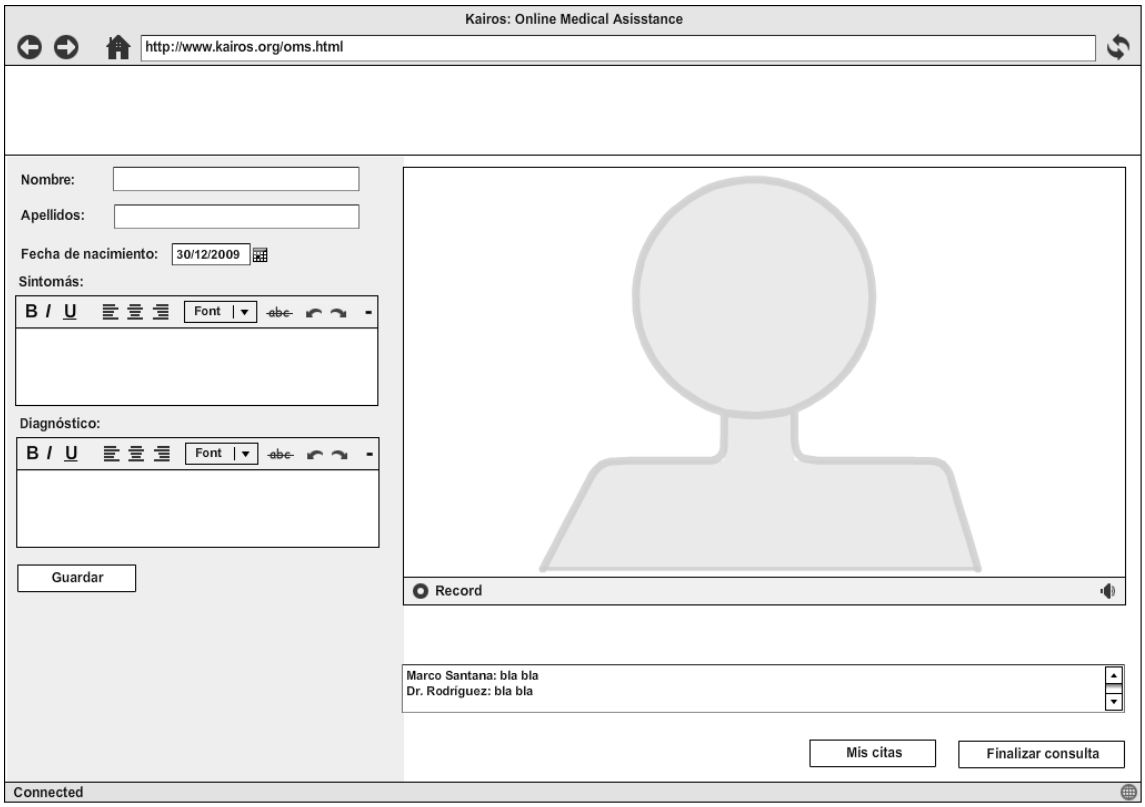
Análisis funcional

Descripción

Durante el proceso de consulta, el médico debe poder ver la información del paciente y actualizarla. La información básica comprenderá el nombre y los apellidos, la fecha de nacimiento, los síntomas y el diagnóstico.

Mockup

Se añadirá un menú lateral a la pantalla de consulta del doctor en el que se mostrará la información del paciente. La información se podrá editar y guardar mediante la adición de un botón. El mockup 7 representa la estructura del detalle de consulta:



Mockup 7: Detalles de consulta

Actores

- Médico.

Historias de usuario

- 7

Restricciones

- El proceso de consulta no puede interrumpirse.
- Los datos mínimos para poder guardar el formulario deberán ser el nombre, los apellidos y la fecha de nacimiento.

Pruebas de aceptación

- Al acceder se muestra la información del paciente ya cargada.
- Al pulsar sobre el botón guardar, la información se actualizará sin interrumpir la comunicación.

Diseño

Durante el proceso de consulta, el doctor puede ir editando la información del paciente. Las acciones que realiza el médico las mostramos en el diagrama de secuencia 7:

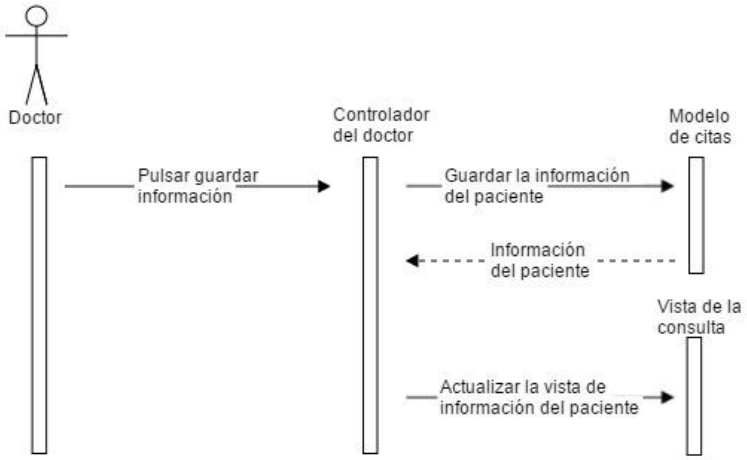


Diagrama de secuencia 7: Guardar información de la consulta

Creamos una nueva vista *vappointment_info*, que integraremos en la vista general *vappointment*. La nueva vista contendrá un formulario que llamará a la función *save_appointment* del controlador mediante Ajax. Al hacerlo así, cargaremos la vista de forma asíncrona y sin interrumpir el proceso de consulta. En el modelo crearemos dos funciones, una para persistir los datos, *save_appointment* y otra para obtener los datos de la consulta, *get_appointment_by_id*. El controlador las utilizará para guardar y obtener los datos de la consulta y mandarlos de nuevo a la vista de información de la consulta. El diagrama de clases 7:

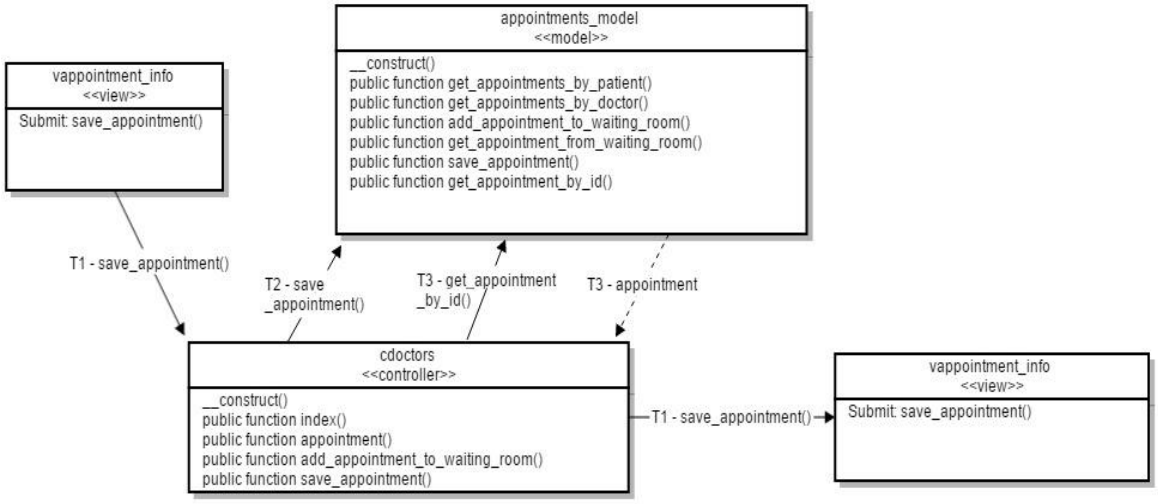
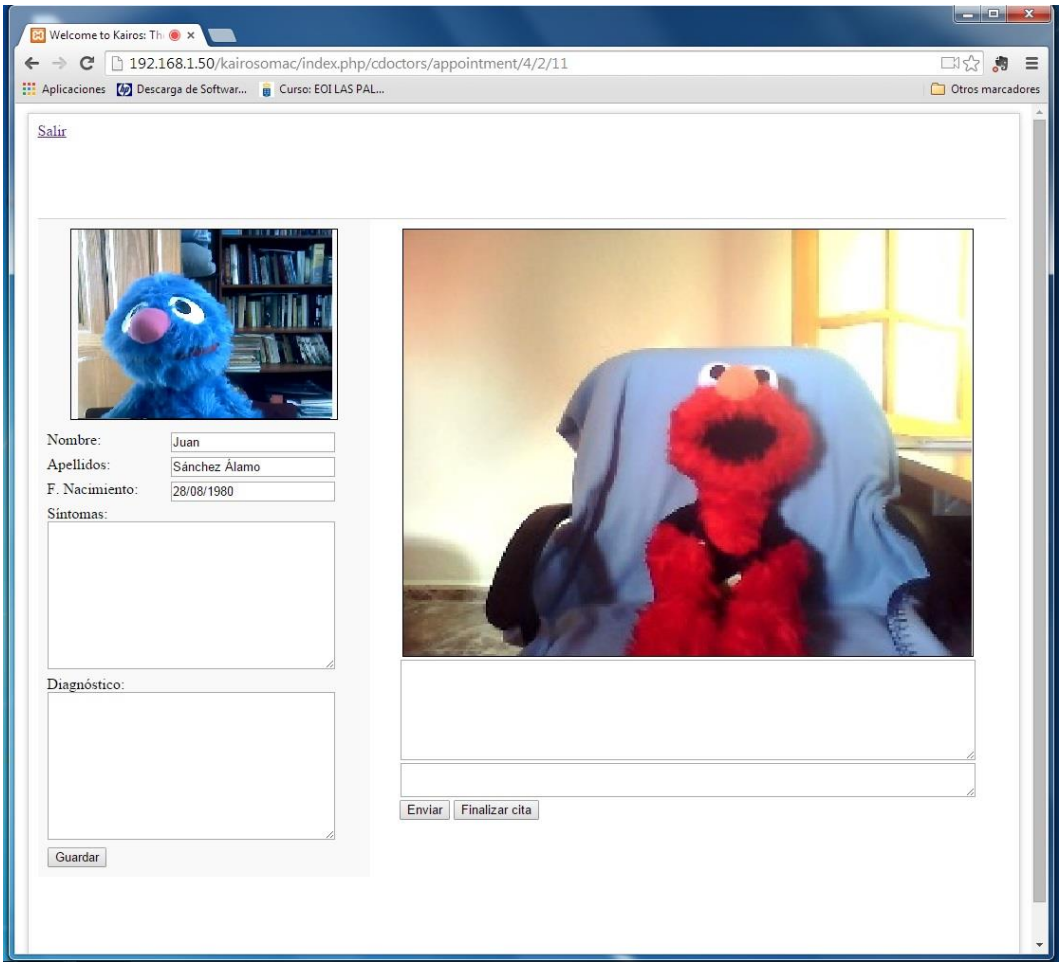


Diagrama de clases 7: Guardar información de la consulta

Pantallas

La pantalla 8 del doctor con la información del paciente y las opciones para modificarlas:



Pantalla 8: Detalles de consulta

8. Resultados y conclusiones

El proceso de desarrollo de un producto software, cómo el de cualquier otra ingeniera, no es un proceso sencillo ni trivial: desde las primeras fases en las que se va conformado la idea de negocio, hasta la obtención del producto, hay que pasar por una serie de etapas en las que cada una supone un riesgo para la consecución de nuestro objetivo final.

Nuestra primera aproximación a la realización del presente proyecto, fue con la idea de implementar una aplicación que se conectara a diversos dispositivos médicos por USB y gestionara toda esa información de forma centralizada. Nos pusimos en contacto con Daniel González Santana, médico especialista en pediatría digestiva del Centro Hospitalario Universitario Insular-Materno Infantil, con la idea de que aceptara el rol de product owner y nos guiara en la realización del mismo. Durante el proceso de extracción de información y desde las primeras entrevistas, fuimos descubriendo de forma conjunta unas necesidades diferentes a las definidas en los objetivos iniciales del proyecto: las urgencias, por normal general, presentaban un número de pacientes significativo y los tiempos de espera no eran cortos. También en los procesos de consultas médicas generales el número de pacientes era elevado. Ambos grupos contaban con un identificador común: en un alto porcentaje, los diagnósticos eran fácilmente realizables mediante una exploración superficial del paciente, ya que eran dolencias bastante comunes: resfriados, sinusitis, fiebres, etc.

Decidimos investigar un poco sobre el tema y descubrimos que en Estados Unidos, el mercado de la teleasistencia sanitaria tenía bastante aceptación, por lo que después de otra serie de entrevistas y conversaciones, decidimos pivotar a este nuevo negocio.

Con la idea del proyecto ya en mente, empezamos a organizar los detalles técnicos para gestionar los riesgos y comprobar la viabilidad del proyecto; el más destacable era la transmisión de video y audio, además de contenido no media, para implementar el módulo de videoconferencia. Afortunadamente investigando este aspecto, dimos con la API WebRTC que nos permitía afrontar el desarrollo de dicha funcionalidad con las garantías de ofrecer un módulo de videconferencia sencillo de integrar, sin necesidad de instalaciones adicionales por parte del cliente, fácil de mantener y con un bajo consumo de recursos.

Con las herramientas ya seleccionadas empezamos el desarrollo del prototipo funcional siguiendo una metodología de desarrollo agíl: análizabamos un hito, lo validábamos con el product owner, realizábamos el diseño técnico, implementábamos la funcionalidad y por último, validábamos el resultado con el product owner, La elección de una metodología ágil venía favorecida en gran medida por la alta disponibilidad de nuestro product owner, permitiéndonos ir de forma conjunta durante el proceso de implementación del prototipo, gestionando sus expectativas y reaccionando rápido a los cambios.

El resultado ha sido bastante satisfactorio, dando como producto un prototipo funcional para la gestión de consultas por videoconferencia. Ademas, podemos destacar la posibilidad de reutilización del módulo de comunicaciones para proyectos de terceros

por haberlo implementado de forma totalmente desacoplada al módulo de gestión de información. También resaltar que la utilización de un ORM para la gestión de la base de datos, nos permite integrar nuestro sistema, con un coste relativamente bajo, en otros sistemas de gestión sanitaria, como un módulo complementario a los servicios originales.

9. Trabajo Futuro.

Como continuidad al trabajo realizado proponemos los siguientes hitos:

- Terminar con la funcionalidad propuesta por el product owner y que no ha sido cubierta en la realización del proyecto presentado.
- Implementación de un módulo de facturación para la gestión del cobro por consultas.
- Implementación y configuración de un servidor NAT/TURN propio para sustituir a los utilizados por el proyecto, que son servidores libres de Google.
- Implementar mecanismos de seguridad en las llamadas por video, como podría ser la aplicación del protocolo HIPAA.
- Implementación de un cliente móvil.
- Implementación de un módulo de personalización de la herramienta en función del cliente.

10. Bibliografía

- [LR14]. Salvatore Loreto & Simon Romano. “Real Time Comunication With Web-RTC”. O’Really, 2014.
- [M02]. Robert C. Martin. “Agile Software Development: Principles, patterns and practice”. Prentice Hall, 2002.
- [M13]. Rob Manson. “Getting Started With WebRTC”. Packt Publishing, 2013.
- [PM08]. Dan Pilone & Russ Miles. “Head First Software Development”. O’Really, 2008.
- [S95]. Ken Schwaber. “SCRUM Development Process”. OOPSLA 1995.

11. Enlaces

- Teladoc:
  - http://www.teladoc.com (12/01/2015)
- MDLive:
  - https://mdlive.com/ (12/01/2015)
- American Well:
  - https://www.americanwell.com/ (12/01/2015)
- Dr On Demand:
  - http://www.doctorondemand.com/ (12/01/2015)
- HIPAA:
  - http://www.hhs.gov/ocr/privacy/hipaa/understanding (12/01/2015)
- ISO 27002:2013:
  - http://www.iso.org/iso/catalogue_detail?csnumber=54533 (12/01/2015)
- VSEE:
  - http://vsee.com/ (12/01/2015)
- AddLive:
  - http://www.addlive.com/ (12/01/2015)
- [O08] Página oficial del creador del modelo canvas:
  - http://alexosterwalder.com (20/01/2015)
- Modelo canvas:
  - http://businessmodelgeneration.com/canvas/bmc (20/01/2015)
- Codeigniter: Framework de desarrollo.
  - https://ellislab.com/codeigniter (10/09/2014)
- Ventajas de Codeigniter:
  - http://www.php-developers.org/blog/codeigniter/advantages-and-features-of-codeigniter-framework.html (10/09/2014)
- Información sobre motores de base de datos:
  - http://db-engines.com/en/ranking (10/09/2014)
- Introducción a los ORMs
  - http://www.orm-designer.com/article/orm-frameworks-for-php5-general-introduction (10/09/2014)

- Información sobre servidores:
  - http://w3techs.com (10/09/2014)

- Propel: ORM.
  - http://propelorm.org/ (10/09/2014)

- Node.js: Servidor Node.
  - http://nodejs.org/ (11/09/2014)

- Socket.io: Librería websocket.
  - http://socket.io/ (11/09/2014)

- Mockflow: Herramienta para los wireframes y mockups.
  - http://www.mockflow.com/ (5/10/2014)

- Gliffy: Herramienta para los diagramas.
  - http://www.gliffy.com/ (5/10/2014)

- Ion Auth: Librería de autentificación para Codeigniter.
  - http://benedmunds.com/ion_auth/ (13/10/2014)

- Bitbucket: servidor Git.
  - https://bitbucket.org/ (6/10/2014)

- Eclipse: entorno de desarrollo.
  - https://www.eclipse.org/ (6/10/2014)

- Git para eclipse: plugin de Git para Eclipse.
  - http://www.eclipse.org/egit/ (6/10/2014)

- Documentación sobre WebTRC
  - http://www.webrtc.org/ (20/01/2015)
  - https://bitbucket.org/webrtc/codelab/ (20/01/2015)
  - http://www.html5rocks.com/en/tutorials/webrtc/basics/ (20/01/2015)
  - http://w3c.github.io/mediacapture-main/getusermedia.html (20/01/2015)
  - http://w3c.github.io/webrtc-pc/#rtcpeerconnection-interface (20/01/2015)
  - http://w3c.github.io/webrtc-pc/#rtcdatachannel (20/01/2015)

- RFC del Protocolo de Descripción de Sesión o SDP
  - https://www.ietf.org/rfc/rfc2327.txt (16/01/2015)

- Draft de JSEP
  - http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03 (16/01/2015)

- Draft del protocolo WebSocket:
  - https://tools.ietf.org/html/rfc6455 (16/01/2015)

Anexos

A. Instalación y configuración del entorno de desarrollo.

Eclipse

Instalaremos y configuraremos primero el Eclipse. Nos hemos bajado la versión Kepler y le hemos añadido el plugin EGit para integrar nuestro repositorio Git, de forma que podamos realizar todas las operaciones desde nuestro IDE. Para instalarlo nos vamos a la opción de instalar nuevo software en el menú 'Help' y añadimos la URL *http://download.eclipse.org/egit/updates* para descargarnos la última versión. Desde el menú "Git repositories" añadimos el repositorio que previamente nos hemos creado en Bitbucket.

Xampp

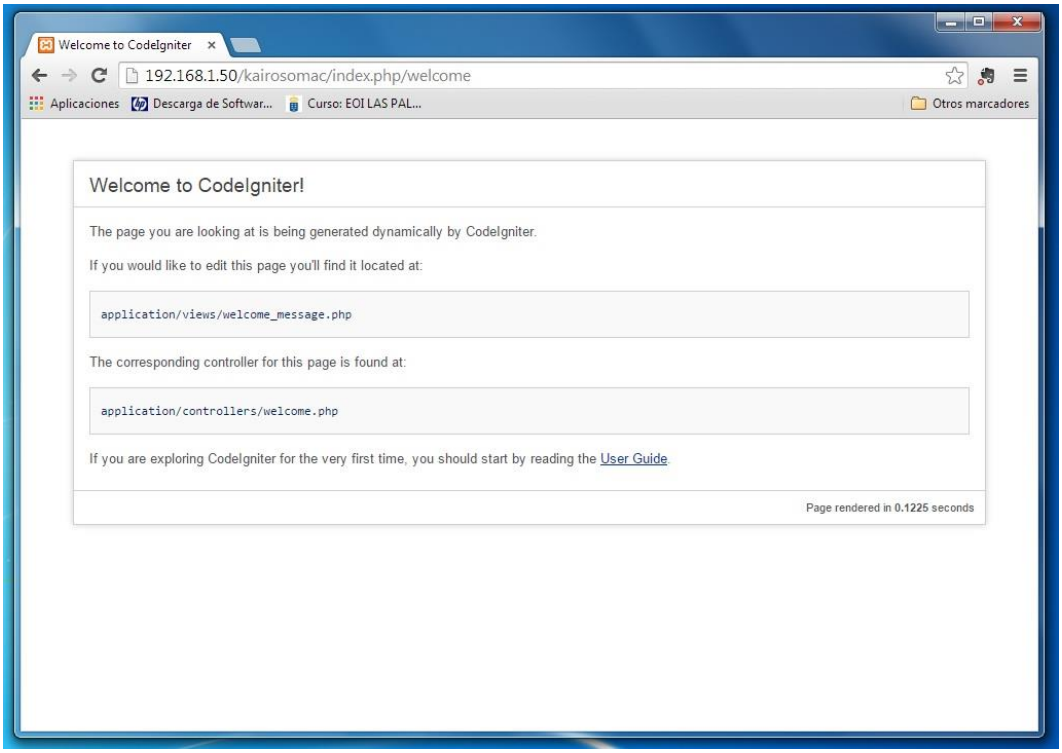
Pasamos ahora a la instalación del servidor XAMPP. XAMPP es una distribución de Apache gratuita y fácil de instalar que contiene MySQL y PHP. La versión disponible en el momento del comienzo del proyecto es la 1.8.1. La instalación es tan sencilla como bajarse el ejecutable e instalarlo. Esto crea una carpeta en la ruta por defecto *C:/xampp* en la que estarán todos los ficheros necesarios para la configuración de nuestro proyecto. Una vez instalado, ya tenemos nuestro servidor Apache y nuestro motor de base de datos MySQL. Además de otras muchas utilidades, la distribución de Xampp viene con PHPMyAdmin, una herramienta gráfica para la gestión de la base de datos MySQL y que nos facilitará mucho la interacción con la misma, y Perl, un intérprete de línea de comandos que nos permite instalar librerías o plugins en nuestro servidor.

Codeigniter

Toca ahora instalar nuestro framework Codeigniter. La instalación es bastante sencilla: nos bajamos la distribución 2.1.4 y la descomprimimos. En el Eclipse nos creamos un proyecto PHP llamado KairosOMAC, importamos el código que acabamos de descomprimir y lo añadimos a nuestro repositorio; esto último creará una carpeta 'git' en nuestro usuario local en donde se ubicará el código que desarrollemos. Para probar que la instalación es correcta creamos en el fichero *C:/xampp/apache/conf/httpd.conf* un alias para poder acceder a nuestra web:

```
Alias /kairosomac "C:/Users/Yeray/git/kairosomac/KairosOMAC"
    <Directory "C:/Users/Yeray/git/kairosomac/KairosOMAC">
      Require all granted
      Header set Access-Control-Allow-Origin "*"
    </Directory>
```

Después de reiniciar el Apache, si accedemos en el navegador a la dirección *http://dirección_ip_local/kairosomac* observaremos la pantalla principal por defecto de Codeigniter:



Pantalla 1

El funcionamiento básico del framework es muy sencillo ya que como hemos comentado anteriormente sigue el patrón MVC: accedemos a un controlador, cargamos librerías, módulos y/o llamadas a modelos para acceder a la base de datos. Desde el controlador llamamos también a las vistas para mostrar la información al usuario por pantalla.

Para acceder a un controlador en Codeigniter, la URL debe seguir el formato siguiente *http://dominio/index.php/controlador/método/parámetro1/parámetro2/...* Omitiendo de la URL las opciones de método y parámetros, accederíamos directamente al método *index* del controlador instanciado. El código que carga la vista anterior al acceder a *http:192.168.1.50/kairosomac/index.php/welcome/index* es:

```
class Welcome extends CI_Controller
{
    public function index()
    {
            $this->load->view('welcome_message');
    }
}
```

Una vez instalado nuestro framework y habiéndonos asegurado que podemos acceder al mismo, configuraremos una plantilla básica para nuestro proyecto que será reutilizada por las diferentes pantallas que iremos creando. Esta vista general estará formada por

una cabecera, una zona para cargar el contenido principal de las distintas pantallas y un pie de página, como vemos en la imagen 1:

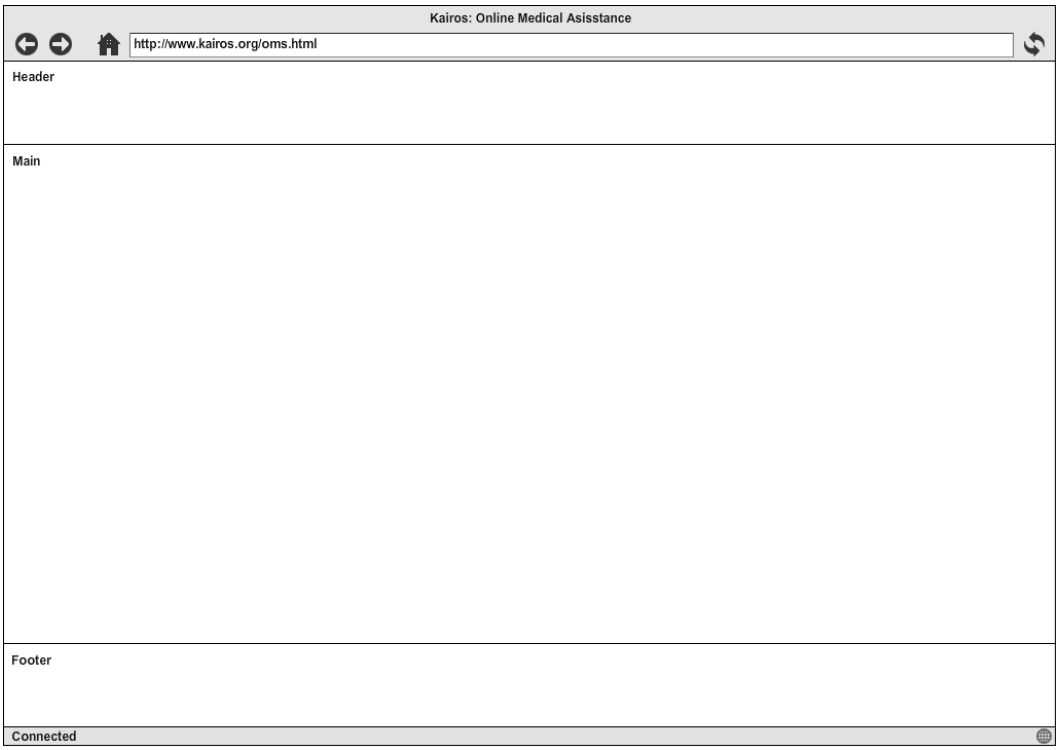


Imagen 1

Por ahora las vistas para la cabecera y el pie de página estarán vacías. El código de la plantilla principal (en *application/views/main/mail_template)* que contendrá a las tres vistas anteriormente mencionadas:

```
<!DOCTYPE HTML>
<html>
    <head>
           <title>
                  Welcome to Kairos: The Online Medical Assistant Service
           </title>
           <meta charset="utf-8" />
    </head>

    <body>
           <div class="page">
                  <div class="page-header">
                     <!-- Vista de la cabecera -->
                     <?php $this->load->view('main/header', $params)
                  </div>

                  <div class="main-section">
                     <!-- Vista del contenido -->
                     <?php $this->load->view($params['main'], $params); ?>
                  </div>

                  <div class="page-footer">
                     <!--  Vista del pie -->
                     <?php $this->load->view('main/footer', $params); ?>
                  </div>
           </div>
    </body>
</html>
```

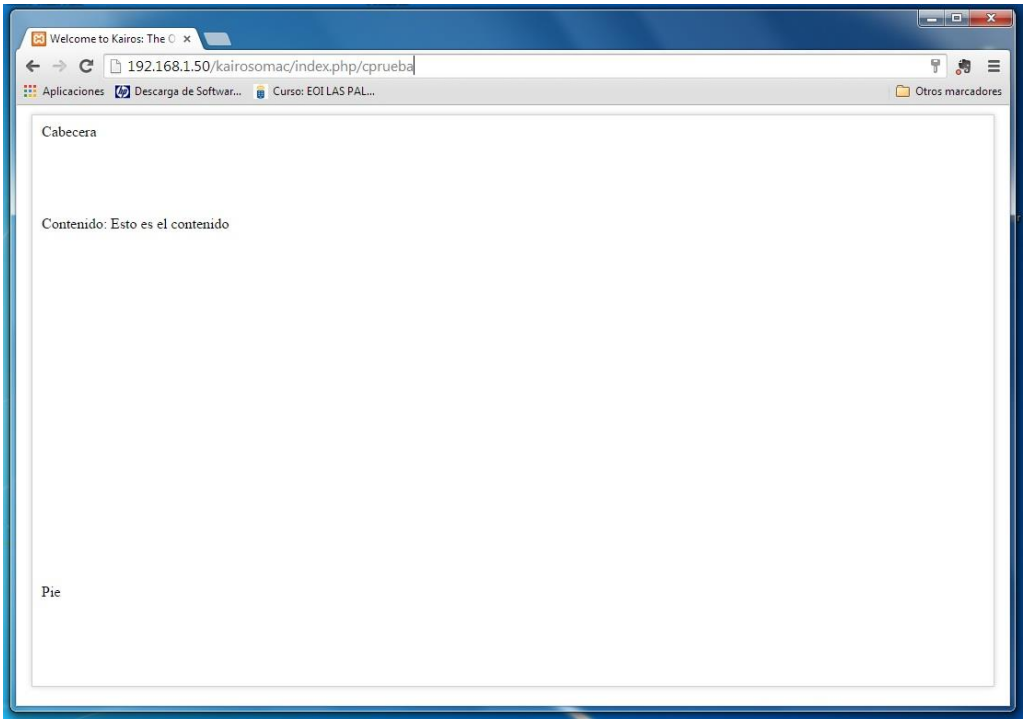
Probaremos ahora la integración completa con un ejemplo básico del funcionamiento del framework. Creamos un controlador *cprueba* dentro de la carpeta */controllers* de Codeigniter con el siguiente método *index*, en el que pasamos una variable a la vista de contenido *vprueba*, vista que cargaremos dentro de la plantilla principal *main_template*:

```
class cprueba extends CI_Controller
{
    public function index()
    {
            // Carga de parámetros para la vista
            $this->data['params']['texto'] = 'Esto es el contenido';

            // Carga de la vista de contenido
            $this->data['params']['main'] = 'prueba/vprueba';

            // Carga de la plantilla principal
            $this->load->view('main/main_template', $this->data);
    }
}
```

Creamos ahora dentro de *application/views/prueba* la vista *vprueba* con el siguiente contenido, que muestra la variable pasada desde el controlador hacia la vista:

```
Contenido: <?php echo $texto; ?>
```

Si accedemos a *http://192.168.1.50/kairosomac/index.php/cprueba* observamos ahora la pantalla 2:



Pantalla 2

Ya tenemos implementado una estructura básica para la gestión de vistas. Vamos ahora a desarrollar una funcionalidad para la gestión de notificaciones al usuario. Para ello utilizaremos el plugin JQuery, UI Notify Widget. Además, crearemos un controlador básico del que heredarán el resto de controladores de nuestra aplicación con la funcionalidad común a todos ellos, como es el caso que nos ocupa. La idea es añadir los mensajes en sesión y luego mostrarlos utilizando el plugin, por lo que al controlador genérico le añadimos tres métodos: uno para gestionar los mensajes, otro las advertencias y por último uno para los errores.

```
class My_Controller extends CI_Controller
{
    function __construct()
    {
           try
           {
                  parent::__construct();
                  error_reporting(E_ALL);

                  // Cargamos el fichero de lenguaje general de la
                  // aplicación
                  $this->lang->load('kairos_common');


           }
           catch (Exception $e)
           {
                  log_message( 'Error', $e->getMessage( ) . ' en ' .
                         $e->getFile() . ':' . $e->getLine() );
           }
    }

    // Método para guardar los mensajes de información
    public function add_message($value)
    {
           if( !empty($value) )
           {
                  $_SESSION['messages'][] = $value;
           }
    }

    // Método para obtener los mensajes de información
    public function get_messages()
    {
           return isset($_SESSION['messages']) ?
                  $_SESSION['messages'] : NULL;
    }

    // Método para borrar los mensajes de información
    public function delete_messages()
    {
           unset($_SESSION['messages']);
    }

    // Método para añadir los mensajes de advertencia
    public function add_warning($value)
    {
           if( !empty($value) )
           {
                  $_SESSION['warnings'][] = $value;
           }
    }

    // Método para obtener los mensajes de advertencia
    public function get_warnings()
    {
```

```
            return isset($_SESSION['warnings']) ?
                  $_SESSION['warnings'] : NULL;
    }

    // Método para borrar los mensajes de advertencia
    public function delete_warnings()
    {
            unset($_SESSION['warnings']);
    }

    // Método para añadir los mensajes de error
    public function add_error($value)
    {
            if( !empty($value) )
            {
                  $_SESSION['errors'][] = $value;
            }
    }

    // Método para obtener los mensajes de error
    public function get_errors()
    {
            return isset($_SESSION['errors']) ?
                  $_SESSION['errors'] : NULL;
    }

    // Método para borrar los mensajes de error
    public function delete_errors()
    {
            unset($_SESSION['errors']);
    }
}
```

Ahora hacemos que nuestro controlador *cprueba* extienda de nuestro nuevo controlador y por lo tanto, herede la funcionalidad del mismo:

```
class cprueba extends My_Controller
{
    public function index()
    {
            $this->data['params']['texto'] = 'Esto es el contenido';

            $this->data['params']['main'] = 'prueba/vprueba';

            // Añadimos un mensaje haciendo uso de la funcionalidad
            // heredada de My_Controller
            $this->add_warning('Sesión finalizada con éxito');

            $this->load->view('main/main_template', $this->data);
    }
}
```

Ya tenemos el mensaje añadido a sesión; vamos ahora a implementar la forma de mostrarlos por pantalla con ayuda del plugin. Lo primero es cargar las librerías de JQuery y la de notificaciones en la plantilla principal.

```
<!DOCTYPE HTML>
<html>

<head>
    .
    .
```

```
    .
    <!-- Cargamos la librería de JQuery -->
    <script type="text/javascript" src="<?php echo base_url
           ('js/jquery-1.9.1.min.js');?>"></script>

    <!-- Cargamos la librería de Notificaciones -->
    <script type="text/javascript" src="<?php echo base_url
           ('js/jquery-notify/ui.notify.js');?>"></script>

    <!-- Cargamos los estilos de la librería de notificaiones -->
    <link rel="stylesheet" type="text/css" href="<?php echo
           base_url('css/ui.notify.css');?>" media="screen" />
    .
    .
    .
</head>
    .
    .
    .

</html>
```

Implementamos una vista para gestionar los mensajes del sistema:

```
<div>

    <!-- Creamos un div con la estructura definida por la librería para
    mostrar las notificaciones -->
    <div id="container" style="display:none">

           <div id="info-notify"
                  class="ui-notify-message ui-notify-message-info">
           <a class="ui-notify-cross ui-notify-close" href="#">x</a>
           <div style="float:left;margin:-7px 10px 0px 0">
           <img src="<?php echo base_url();?>
                  /css/images/information.png" alt="info"></div>
           <h1><?php echo $this->lang->line('information');?></h1>
           <p>#{text}</p>
           </div>

           <div id="warning-notify">
           <a class="ui-notify-cross ui-notify-close" href="#">x</a>
           <div style="float:left;margin:-7px 10px 0 0">
           <img src="<?php echo base_url();?>
                  /css/images/warning.png" alt="warning"></div>
           <h1><?php echo $this->lang->line('warning');?></h1>
           <p>#{text}</p>
           </div>

           <div id="error-notify">
           <a class="ui-notify-cross ui-notify-close" href="#">x</a>
           <div style="float:left;margin:-7px 10px 0 0">
           <img src="<?php echo base_url();?>
                  /css/images/error.png" alt="error"></div>
           <h1><?php echo $this->lang->line('error');?></h1>
           <p>#{text}</p>
           </div>
    </div>

    <?php
           // Cargamos los mensajes de información que se han
           // añadido
           $messages = get_instance()->get_messages();

           // Añadimos al DOM los mensajes de información
```

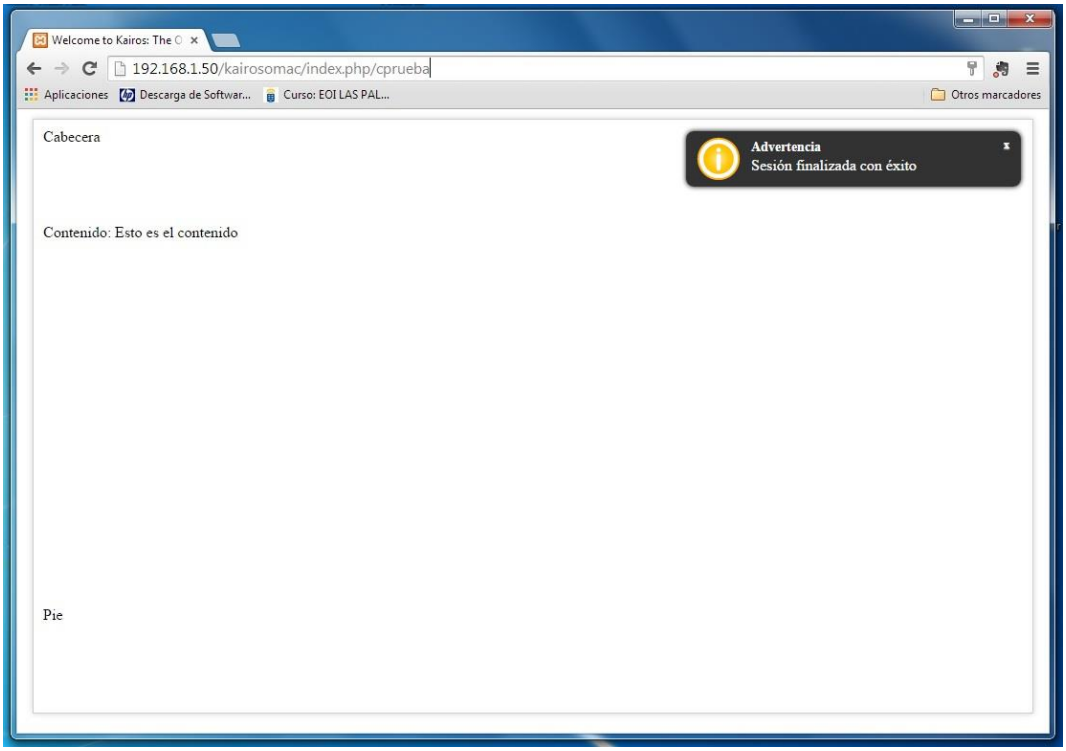
```
            if(isset($messages))  { ?>
                  <div class="messages" style="display:none">
                        <ul>
                              <?php  foreach( $messages as $message )
                              {
                                    echo sprint
                                          ('<li>%s</li>', $message);
                              } ?>
                        </ul>
                  </div>
            <?php
                  // Borramos los mensajes
                  get_instance()->delete_messages();
            }
      ?>

      <?php
            // Cargamos los mensajes de advertencia que se han
            // añadido
            $messages = get_instance()->get_warnings();

            // Añadimos al DOM los mensajes de advertencia
            if(isset($messages))  { ?>
                  <div class="warnings" style="display:none">
                        <ul>
                              <?php  foreach( $messages as $message )
                              {
                                    echo sprint
                                          ('<li>%s</li>', $message);
                              } ?>
                        </ul>
                  </div>
            <?php
                  // Borramos los mensajes
                  get_instance()->delete_warnings();
            }
      ?>

      <?php
            // Cargamos los mensajes de error que se han
            // añadido
            $messages = get_instance()->get_errors();

            // Añadimos al DOM los mensajes de error
            if(isset($messages))  { ?>
                  <div class="errors" style="display:none">
                        <ul>
                              <?php  foreach( $messages as $message )
                              {
                                    echo sprint
                                          ('<li>%s</li>', $message);
                              } ?>
                        </ul>
                  </div>
            <?php
                  // Borramos los mensajes
                  get_instance()->delete_errors();
            }
      ?>

</div>



<script>

      $(function()
      {
```

```
            // Añadimos la funcionalidad.
            $("#container").notify();

            // Por cada mensaje añadido al DOM, mostramos una notificación
            // haciendo uso de la librería
            $.each( $('.messages > ul > li'), function(id, value)
            {
                  $("#container").notify("create", "info-notify", {
                        text: $(value).html()
                  });

            });

            $.each( $('.warnings > ul > li'), function(id, value)
            {
                  $("#container").notify("create", "warning-notify", {
                        text: $(value).html()
                  });

            });

            $.each( $('.errors > ul > li'), function(id, value)
            {
                  $("#container").notify("create", "error-notify", {
                        text: $(value).html()
                  });
            });

</script>
```

Y la añadimos a la plantilla principal:

```
<!DOCTYPE HTML>
<html>

<head>
    .
    .
    .
</head>

<body>

    <div class="page">

            <div class="page-header">
                  <?php $this->load->view('main/header', $params); ?>
            </div>

            <div class="main-section">
                  <?php $this->load->view($params['main'], $params); ?>
            </div>

            <div class="page-footer">
                  <?php $this->load->view('main/footer', $params); ?>
            </div>

    </div>

    <!-- Añadimos la vista de notificaciones -->
    <?php $this->load->view('main/notify', $params); ?>

</body>

</html>
```

Ejecutando la aplicación, observamos lo que podemos ver en la pantalla 3:



Pantalla 3

Propel

Una vez configurado nuestro framework pasamos a instalar Propel. Podemos instalar Propel de dos formas distintas: la primera es dentro del proyecto en cuestión y la segunda es a través de Pear. La diferencia principal entre una y otra es que si instalamos Propel de la primera manera, el ORM sólo estará disponible para el proyecto en cuestión. Sin embargo, si la instalamos de la segunda forma, tendremos instalado el ORM de forma general para todos los proyectos. Elegimos instalarlo de la primera manera y así hacemos independiente la versión del ORM entre los distintos proyectos que queramos desarrollar. Los pasos para instalar Propel son los siguientes:

1. Nos descargamos el ORM, en nuestro caso la versión 1.7.
2. Lo descomprimimos dentro de una carpeta en nuestro proyecto, *application/third_party*.
3. Instalamos la dependencia Phing a través de Pear. Para ello ejecutamos en la línea de comandos (ejecutarla con permisos de administrador) la siguiente cadena de comandos. Recordar que la ruta de los .bat debe estar añadida como variable del sistema en el PATH de Windows o tendremos que ir a la ubicación donde se encuentran para poder ejecutarlos (*C:\xampp\php*). Esto lo tendremos que hacer para que Propel puede ejecutar Phing. Una vez instalado tecleamos en la línea de comandos *phing -v* para comprobar que se ha instalado correctamente:

   a. pear channel-discover pear.phing.info
   b. pear install phing/phing
   c. pear install Log
4. Probamos la instalación. Vamos a la carpeta donde hemos instalado la librería Propel y en /propel/generator/bin/ ejecutamos el comando propel-gen. Se empezará a ejecutar el script mostrando un mensaje de fallo. Esto es normal ya que no hemos configurado la librería.

Configuraremos ahora nuestro ORM. Lo primero es decirle a Propel que sistema de gestión de base de datos estamos utilizando y cuál es el nombre de la misma; eso lo hacemos en el fichero *application/third_party/propel-1.7.0/generator/bin/build build.properties*.

```
# Database driver
propel.database = mysql

# Project name
propel.project = kairos
```

Propel necesita comunicar en tiempo de ejecución el modelo que generemos de los objetos con la bbdd. Para ello hace uso de un fichero de configuración, *application/third_party/propel-1.7.0/generator/bin/build/runtime-conf.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <propel>
           <datasources default="kairos">
           <datasource id="kairos">

                  <adapter>mysql</adapter>

                  <connection>

                         <dsn>mysql:host=localhost;dbname=kairos</dsn>

                         <user>kairos</user>

                         <password>kairos</password>

                         <settings>
                                <setting id="charset">utf8</setting>
                         </settings>

                         <options>
                                <option id="MYSQL_ATTR_INIT_COMMAND">
                                       SET NAMES utf8 COLLATE
                                       utf8_unicode_ci
                                </option>
                         </options>

                  </connection>

                  </datasource>
           </datasources>
    </propel>
</config>
```

Configuramos en dicho fichero las opciones de acceso a nuestra base de datos (base de datos que previamente hemos creado con ayuda de PHPMyAdmin). Es el momento ahora para crear un objeto propel y modelar una tabla de nuestra base de datos; crearemos la tabla usuarios y la modelaremos. En principio la tabla sólo contendrá un identificador y un campo nombre. El modelado de las tablas se hace en el fichero *application/third_party/propel-1.7.0/generator/bin/build/schema.xml*, cada tabla se define dentro de las etiquetas *<table></table>*. La definición de la tabla usuarios:

```
<?xml version="1.0" encoding="utf-8"?>

<database name="kairos" defaultIdMethod="native">

    <table name="users" phpName="Users">
           <column name="id" type="integer" required="true"
                  primaryKey="true" autoIncrement="true"/>
           <column name="name" type="varchar" size="50" />
    </table>

</database>
```

Y la definición de nuestra tabla usando MySQL:

```
CREATE TABLE IF NOT EXISTS `users` (
    `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
    `name` varchar(100) CHARACTER SET utf8 COLLATE utf8_spanish2_ci
           DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci
    AUTO_INCREMENT=2 ;
```

Para más información sobre cómo definir las tablas en términos de nuestro ORM nos remitiremos a la documentación de Propel: ahí se explica bien la correspondencia de campos entre Propel y nuestra base de datos. Construiremos ahora el modelo mediante el comando *propel-gen om* en la ruta *application/third_party/propel-1.7.0/generator /bin*. Esto creará un nuevo directorio *application/third_party/propel-1.7.0/generator /bin/build/classes/kairos* con las clases Propel. Por cada tabla, Propel genera 3 clases:

- La clase modelo, que representa una fila dentro de la bbdd.
- La clase peer, que ofrece las constantes y métodos para operar con el objeto.
- La clase query, que se utiliza para operar con la tabla.

Ahora relacionaremos los objetos de la base de datos con las clases PHP generadas, ejecutando el comando *propel-gen convert-conf*. Ya tenemos Propel instalado y listo para ser utilizado. Probaremos ahora que todo está correcto. Modificamos el método *index* de nuestro controlador *cprueba*, nos creamos un modelo al que llamaremos *prueba_model*, cargamos los datos de la tabla *users* y lo devolveremos al controlador para que los muestre en la vista *vprueba*.

```
class cprueba extends CI_Controller
{

    public function index()
    {
           // Cargamos el modelo
           $this->load->model('uers_model');
```

```
            // Llamamos al modelo para obtener los usuarios
            $this->data['params']['users'] =
                  $this->users_model->get_users();

            // Establecemos la vista
            $this->data['params']['main'] = 'prueba/vprueba';

            // Cargamos la vista en la plantilla principal y
            // la mostramos
            $this->load->view('main/main_template', $this->data);
    }

}
```

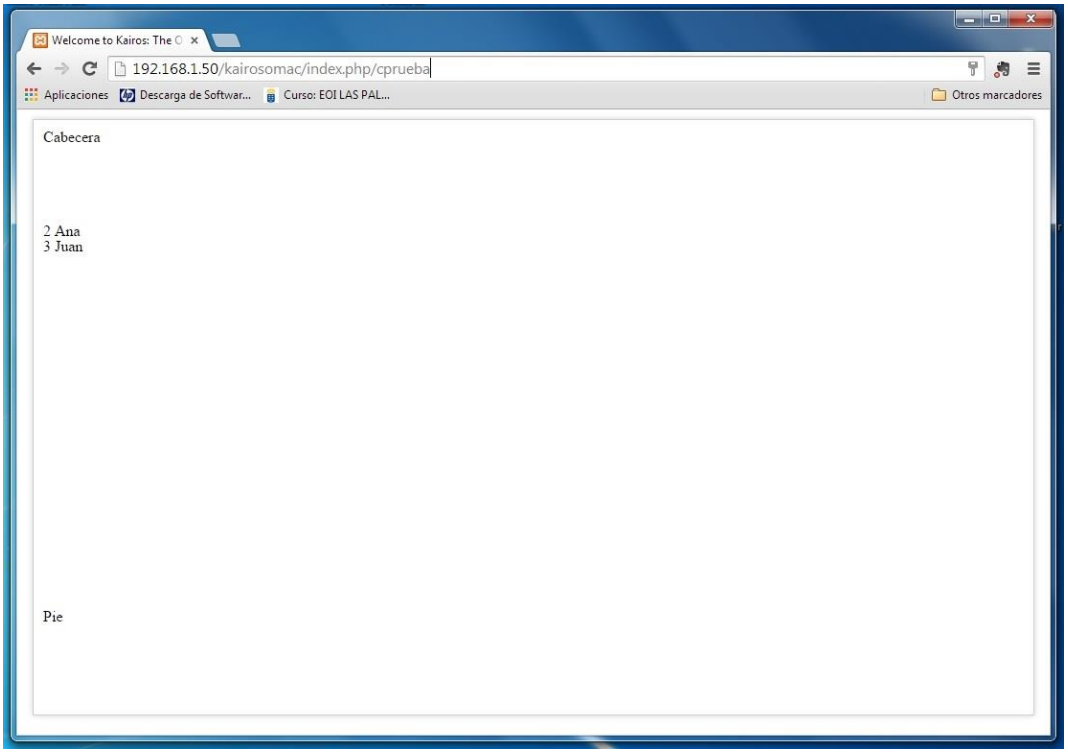
En el modelo crearemos un constructor en el que configuraremos el acceso a Propel para poder utilizarlo y un método *get_users()* para obtener los usuarios haciendo uso de nuestro ORM:

```
class Users_model extends CI_Model {

    public function __construct()
    {
            parent::__construct();

            // Incluímos el script de Propel
            require_once 'application/third_party/
                  propel-1.7.0/runtime/lib/Propel.php';

            // Inicializamos Propel
            Propel::init("application//third_party/
                  propel-1.7.0/generator/bin/build/conf/kairos-conf.php");

            // Añadimos el directorio con las clases PHP que
            // modelan a los objetos de la base de datos
            set_include_path("application//third_party/
                  propel-1.7.0/generator/bin/build/classes"
                  . PATH_SEPARATOR . get_include_path());
    }


    public function get_users()
    {
            // Creamos la consulta para obtener los usuarios
            $users_query =  UsersQuery::create()->find();

            $users = array();
            foreach ($users_query as $user_query)
            {
                  $user['id'] = $user_query->getId();
                  $user['name'] = $user_query->getName();

                  $users[] = $user;
            }

            return $users;
    }

}
```

En la vista mostramos los datos obtenidos por la consulta:

```
<div>
    <ul>
           <?php foreach ($users => $user) { ?>
                  <li>
                         <?php
                                echo $user['id'] . ' ' .  $user['name'];
                         ?>
                  </li>
           <?php } ?>
    </ul>
</div>
```

Si nos vamos a la dirección *http://direccion_ip_local/kairosomac/index.php/cprueba* en nuestro navegador, comprobamos que accedemos a nuestra BBDD utilizando Propel como observamos en la pantalla 4:



Pantalla 4

Node.js

Por último instalaremos nuestro servidor Node. La instalación de Node.js en entornos Windows es bastante sencilla: basta con bajarnos desde su página web oficial la versión que queramos (0.10 en nuestro proyecto) y lanzar el ejecutable. Una vez instalado (y configuradas las variables de entorno correctamente) para acceder al mismo solamente tenemos que teclear *node* en el bash de Windows. Además, Node viene con una utilidad de línea de comandos para la compilación, instalación y actualización de módulos así como la gestión de las dependencias en nuestro servidor: NPM. Instalar una es tan sencillo como ejecutar la siguiente línea, una vez estemos en el bash de Node:

```
npm install "nombre_librería"
```

Con todo instalado, empezamos a crear nuestra aplicación para la gestión de comunicaciones mediante audio, video y chat. En el Eclipse creamos un proyecto Javascript que contendrá el proyecto KairosOVC. Repetiremos los mismos pasos que para nuestro proyecto KairosOMAC para integrarlo con nuestro repositorio en BitBucket. Una vez hecho esto, lo que debemos hacer es configurar nuestra aplicación para que sirva contenido estático y por lo tanto atienda a las peticiones de otros proyectos cuando se le soliciten ficheros; así los proyectos externos podrán incorporar como librería externa nuestro sistema de comunicación. Para ello instalaremos el paquete node-static; este paquete da soporte a peticiones Get y Head condicionales.

```
npm install node-static
```

Una vez instalado, creamos un fichero en la raíz del proyecto con el nombre de server.js. Este será el fichero principal de nuestro servidor y en el que implementaremos la funcionalidad para que nuestro servidor pueda recibir peticiones de ficheros y atenderlas. Lo primero es cargar la librería:

```
var static = require('node-static');
```

Una vez instalada, ya podemos crear una instancia de la misma pasándole como parámetro la ruta relativa en la que se encontrarán los ficheros a servir. En nuestro servidor, crearemos una carpeta public en la que alojaremos los ficheros de la librería que queremos que sean accesibles:

```
var file = new static.Server('./public');
```

El resto de la implementación es muy sencillo: ante una petición http de un fichero mediante la instrucción request, servimos el fichero solicitado con la opción serve de la instancia de la librería:

```
require('http').createServer(function (request, response) {
    request.addListener('end', function () {
        file.serve(request, response);
    }).resume();
}).listen(2013);
```

El código un poco más elaborado, con captura de errores a la hora de servir ficheros queda:

```
var static = require('node-static');
var file = new static.Server('./public');
var http = require('http');

var app = http.createServer(function (request, response) {
    request.addListener('end', function () {
        file.serve(request, response, function (err, result) {
```

```
            if (err) { // Error sirviendo el fichero
                 if(err.status === 404) {
                        // Fichero no encontrado
                        sys.error("Error 404: File " + request.url +
                        " not found ");
                        response.writeHead(err.status, err.headers);
                        response.end();
                 }
                 else {
                        // Otros errores
                        sys.error("Error serving " + request.url + "
                        - " + err.message);
                        response.writeHead(err.status, err.headers);
                        response.end();
                 }
           }
       });
    }).resume();
}).listen(2013);
```

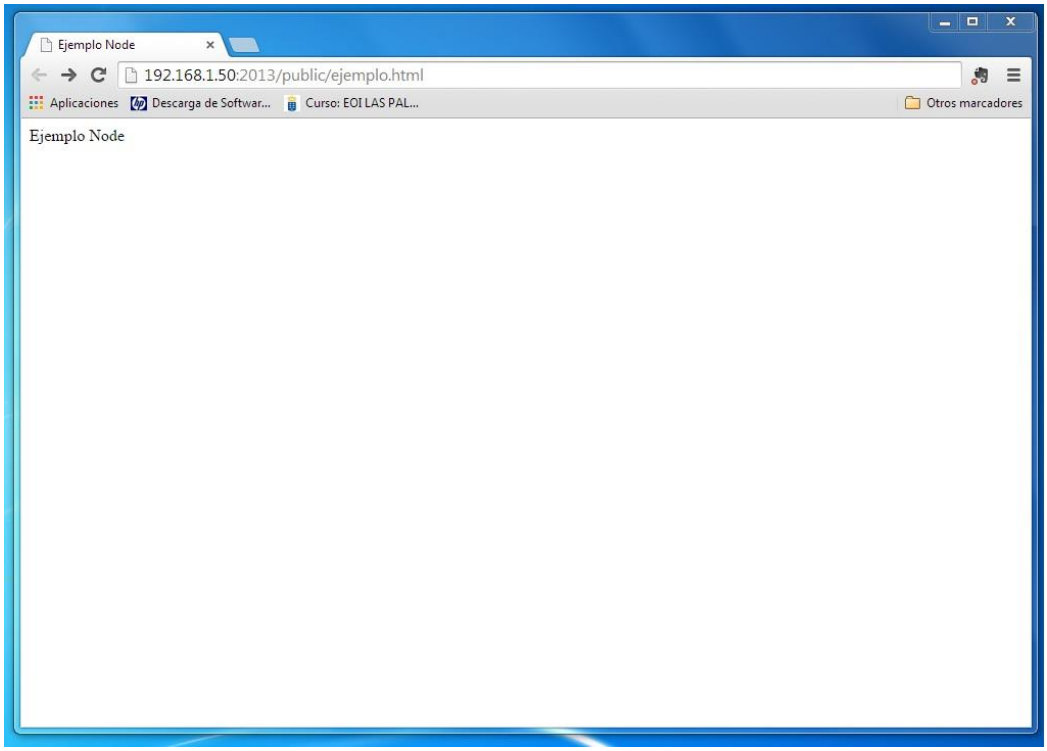
Para ejecutar la aplicación nos situamos en la carpeta del proyecto y ejecutamos en el intérprete de comandos de Node la sentencia:

```
node server.js
```

Ahora podemos servir páginas web estáticas. Añadimos a la carpeta *public* el siguiente HTML y con el Node ejecutándose accedemos a la dirección *http://node_server_ip :node_server_port/public/ejemplo.html*:

```
<html>
    <head>
           <title>Ejemplo Node</title>
    </head>
    <body>
           Ejemplo Node
    </body>
</html>
```

El resultado en la pantalla 5:



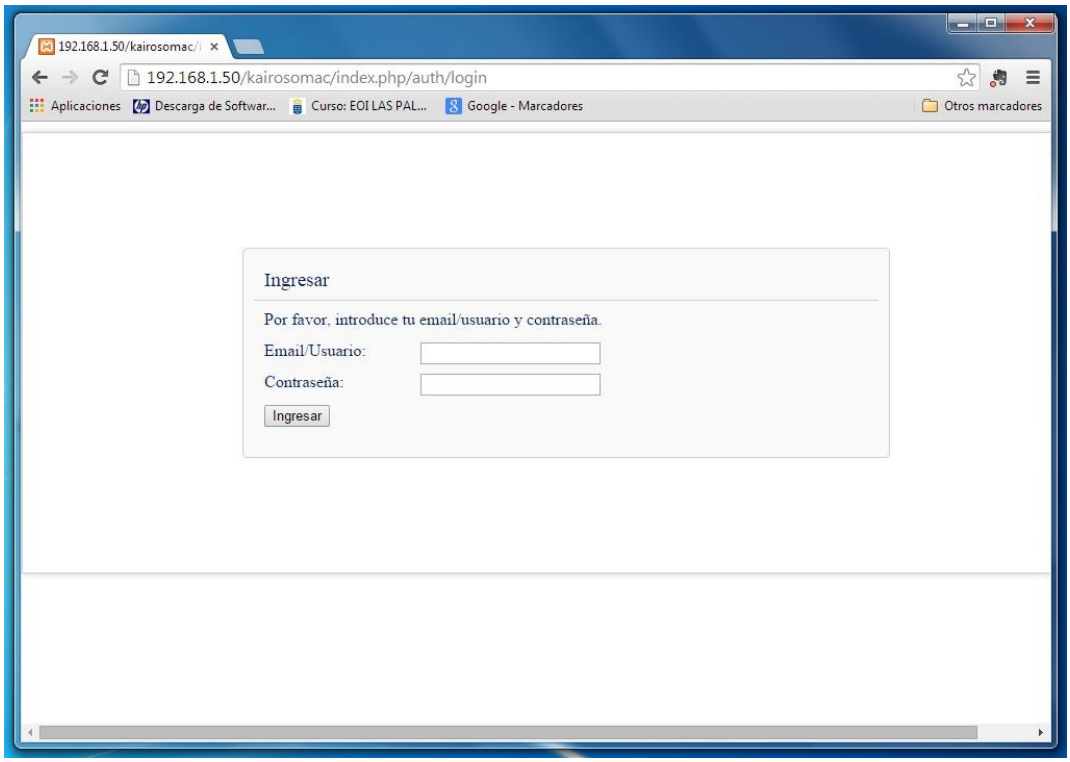
Pantalla 5

También podemos añadir ficheros a proyectos web externos bajo demanda, lo que nos permitirá que éstos incluyan nuestras librerías y por ende que hagan uso de la funcionalidad que queramos publicar:

```
<script src='http://node_server_ip:node_server_port/public/file.js'>
</script>
```

B. Manual de usuario

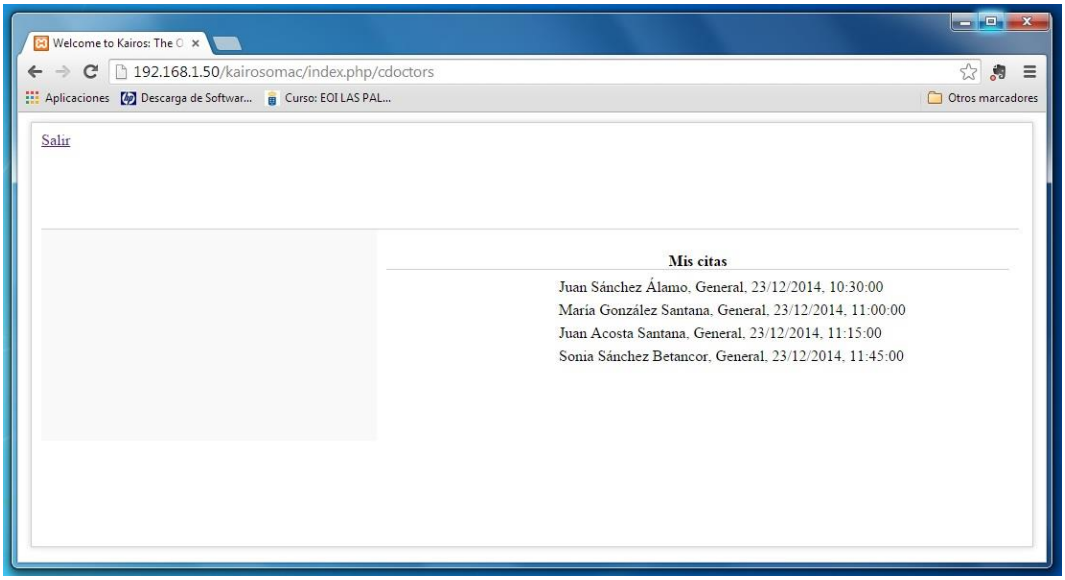
Acceso a la aplicación



Pantalla 1: Acceso a la aplicación

Para poder acceder a la aplicación, el usuario debe ponerse en contacto con el equipo de soporte para darse de alta mediante una combinación email/contraseña que deberá introducir en el formulario de la pantalla 1. Una vez establecidos dichos valores, el usuario podrá acceder a la misma, siendo redirigido a la pantalla principal de gestión del usuario.

Pantalla principal del paciente



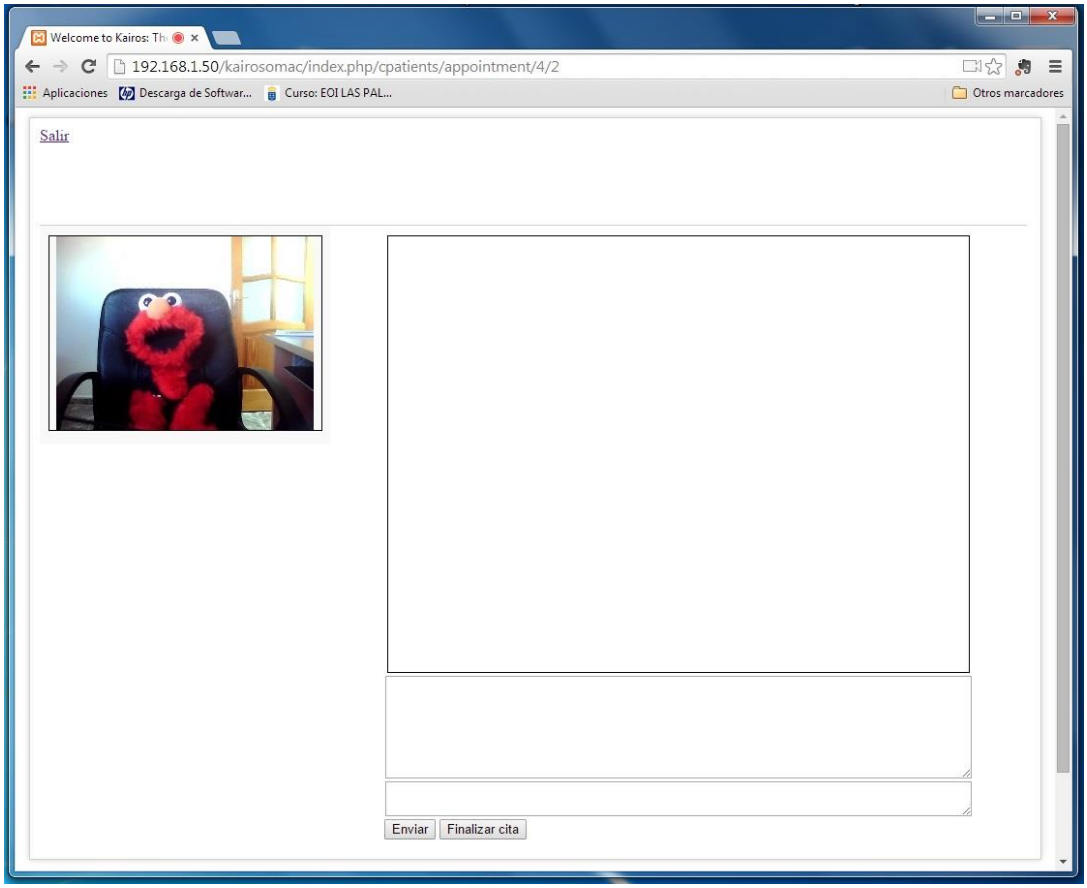
Pantalla 2: Pantalla de gestión del paciente

La pantalla 2 muestra la pantalla principal del paciente, desde la que tendrá acceso a la funcionalidad del sistema. Desde aquí, el paciente podrá ver un listado con las citas futuras y acceder a una consulta.

Para que el paciente pueda ser atendido por un facultativo, deberá pulsar sobre el enlace activo de la consulta a la que va a acceder.

Cuando el paciente pulsa sobre la cita, se redirigirá a la pantalla de consulta, permaneciendo a la espera hasta el acceso del facultativo.

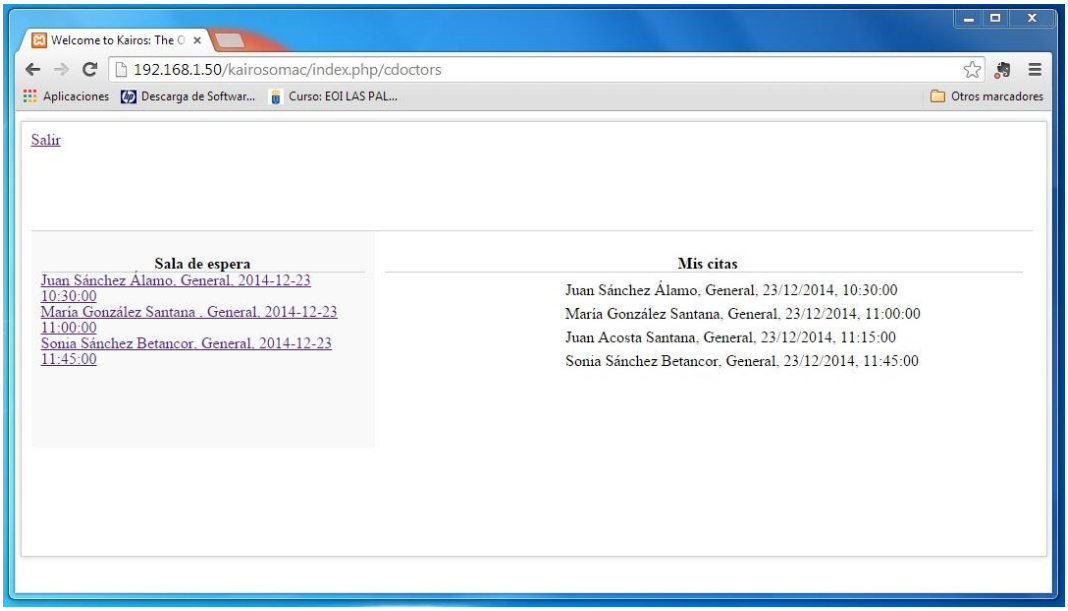
Pantalla de consulta del paciente



Pantalla 3: Consulta del paciente

En esta pantalla, la número 3, el paciente recibirá la consulta por video conferencia. Además tendrá la opción de utilizar un chat. Para concluir la cita, el paciente podrá pulsar sobre el botón de “Finalizar cita”, lo que le devolverá a la pantalla principal.

Pantalla principal del facultativo

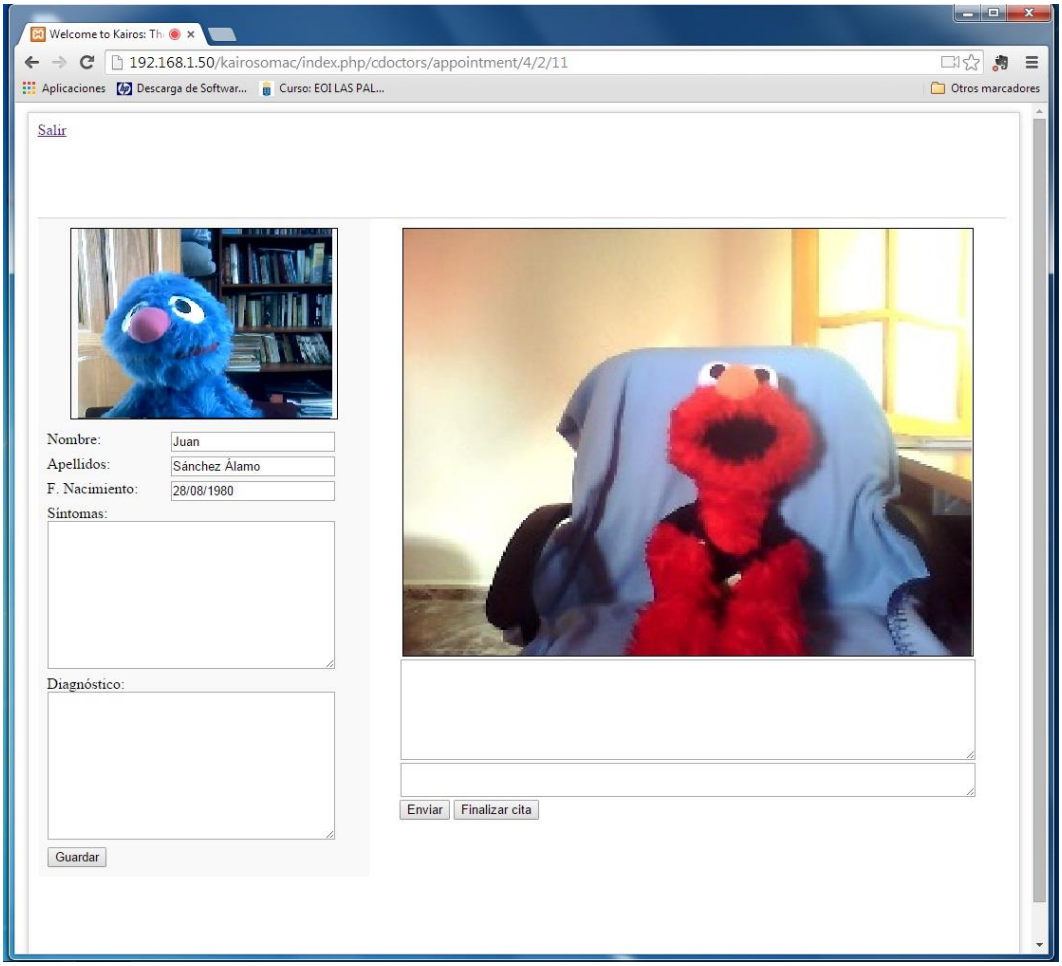


Pantalla 4: Gestión del facultativo

La pantalla 4 es la pantalla principal del facultativo. En el apartado central, "Mis citas", se muestra un listado con las citas del día actual.

En el menú de la izquierda, "Sala de espera" se irán introduciendo las citas de los pacientes que estén en espera. Para acceder a una sesión de consulta, el doctor sólo tiene que pulsar sobre una de estas citas y será redirigido a la pantalla de consulta, en la que estará esperando el paciente.

Pantalla de consulta del facultativo



Pantalla 5: Consulta del facultativo

Desde la pantalla 5, el doctor atenderá al paciente mediante videoconferencia y la posibilidad de chat. Podrá consultar la información básica del paciente y guardar la información de la consulta.

Una vez haya terminado la cita, el facultativo podrá volver a la pantalla principal mediante el botón de finalizar cita.