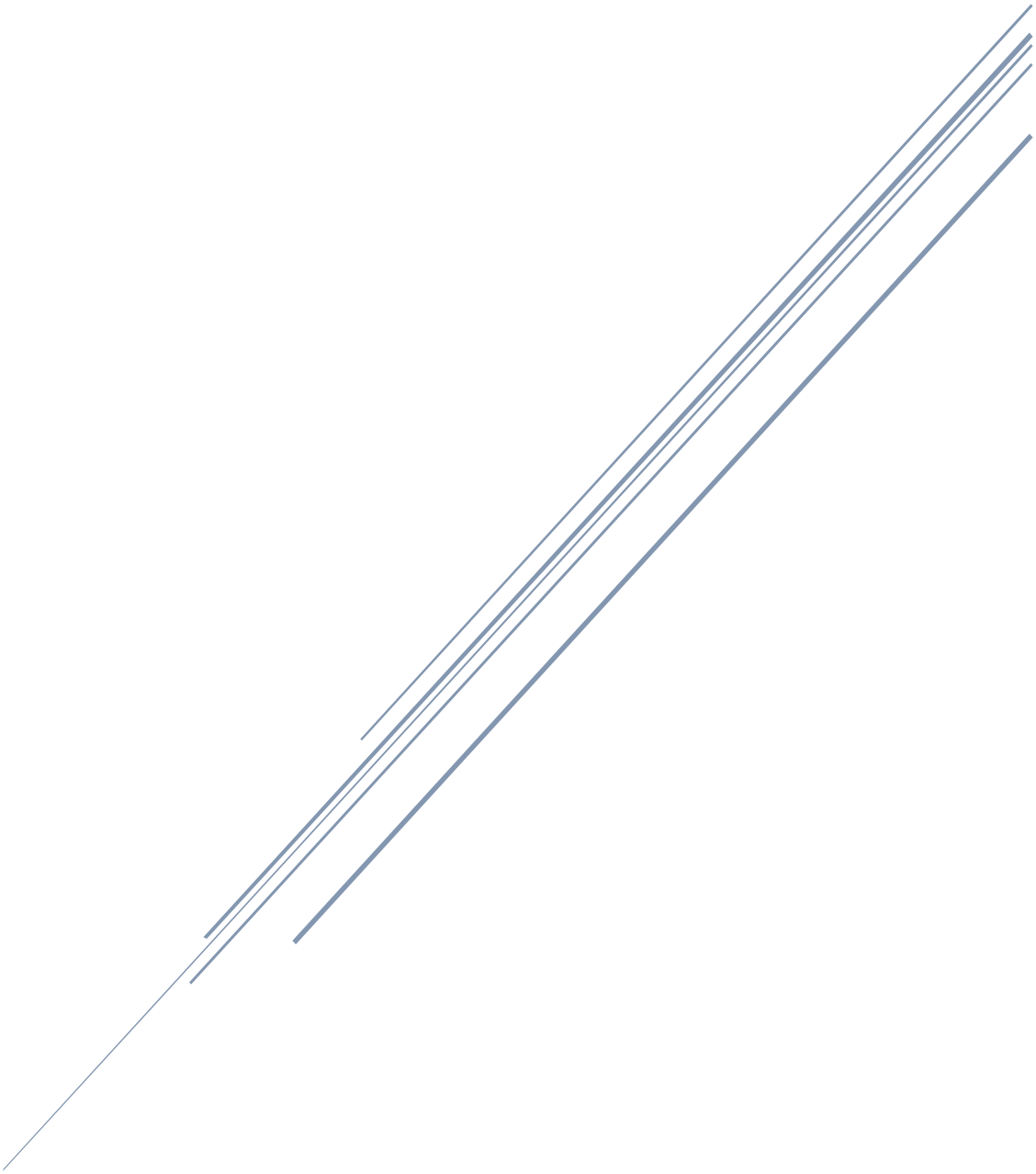
PRÁCTICA CHAT SEGURO

Seguridad en el diseño de software



Encarna Amorós Beneite y Jorge Azorín Martí

06/06/2016

Índice

Información del proyecto

Nombre proyecto

El nombre del proyecto a realizar es el de securityChat. Se trata principalmente de un chat seguro.

Miembros autorizados

Los miembros autorizados en el proyecto son Encarna Amorós Beneite y Jorge Azorín Martí.

Confidencialidad, integridad y disponibilidad

Las conversaciones que los usuarios tengas en el chat, así como sus datos personales serán totalmente confidenciales. No serán distribuidos ni se comercializará con ellos. La aplicación proporcionará los datos de forma íntegra, sin errores o con un formato sin posibilidad de ser entendido por el servidor o atacantes (datos codificados, etc.) La aplicación trata de ser tolerante a fallos, es decir, es difícil que el usuario pueda introducir algún dato incorrecto que provoque una excepción de la ejecución. Y en el caso de haber algún problema, el flujo no se detiene pero se notifica al usuario de lo ocurrido. Cuando el sistema entre en producción, el servicio estará disponible las 24 horas.

Estado operacional software

El estado operacional de la aplicación en la actualidad es de un software listo para poner en producción. Necesitaríamos un servidor de bases de datos y un dominio para que los usuarios puedan utilizar la aplicación vía web. Además, la interfaz seria reutilizable para sin problemas, crear una aplicación híbrida para android o apple. Y debido a que el rendimiento es mejor en aplicaciones nativas, en un futuro se crearía la aplicación nativa para android y apple.

Importancia para organización

Con este proyecto planeamos que usuarios con una necesidad urgente de privacidad a la hora de hablar con otros usuarios, puedan encontrar lo que buscan con la utilización de este chat seguro.

Descripción general del valor o cometido del software

El chat cuenta con un valor añadido que no poseen las aplicaciones de chat hoy en día más utilizadas. Este valor añadido es la seguridad que porta, la cual explicaremos en profundidad en los siguientes apartados. Y como ya hemos dicho la confidencialidad de los datos. Ni siquiera el servidor conoce los mensajes o claves sin cifrar. Y aunque un atacante consiguiente robar la base de datos, no podría descifrar ningún mensaje o clave del usuario.

En resumen el software cuenta con las siguientes características:

- **Cifrado en tiempo real**: El chat es interactivo y sin latencias.
- **Mecanismo de autenticación seguro**: gestión de contraseñas utilizando SHA2 y Bcrypt.
- **Transporte de red seguro**: La comunicación entre el cliente y el servidor es vía TLS y la comunicación entre el cliente y el cliente para el navegador en Angular JS es vía HTTPS. Además se utilizan certificados asegurando la integridad de los mensajes.
- **Optimización de la privacidad**: El servidor solo recibirá datos cifrados del cliente.
- **Chats en grupo**: Se pueden crear chats añadiendo a más usuarios a un chat, eliminarlos y cambiar el nombre del chat. También hay un buscador para encontrar rápidamente un chat por su nombre.
- **Directorio de usuarios**: Los usuarios tienen un perfil el cual pueden modificar. Se pueden realizar búsquedas de usuarios y ver sus perfiles.
- **Chat offline**: Si un usuario no se encuentra online en el momento en el que le envían mensajes, estos les aparecerá como aviso o notificación, cuando vuelva a estar online.
- **Multidispositivo** (mantiene el estado): Aunque cambie de dispositivo, el usuario iniciando sesión, podrá utilizar la aplicación con normalidad.
- **Interfaz web**: La interfaz para que el usuario utilice la aplicación se desarrollará no en consola, si no con una interfaz web.

Objetivos

Confidencialidad

En cuanto a la filtración de información como ya se ha comentado no se publicará ni se dará la información de ningún usuario, como datos, claves, mensajes… Las medidas de seguridad a tomar en cuanto a la confidencialidad son las siguientes:

Para la confidencialidad de los mensajes: Se requiere que los mensajes de cada chat se encuentren cifrados a la hora de almacenarlos o comunicarlos (de servidor a cliente y entre los clientes). Esto hace que el servidor no conozca ningún mensaje pues están cifrados y tampoco tiene la capacidad para poder descifrar dichos datos (mensajes de chats, claves del usuario, etc.) Se debe controlar que los mensajes sigan siendo confidenciales aún respecto a usuarios que estuvieron en un chat y ya no están.

Transporte de red: Este transporte debe estar protegido con algún protocolo seguro, de tal forma que se proteja la confidencialidad de los mensajes.

Memoria o caché: Cuando un usuario que ha iniciado sesión, cierre la ventana del navegador donde está interactuando con la web, se dejará tener constancia de sus datos. Estos datos no serán guardados en ninguna base de datos del cliente, cookies, etc. El usuario la próxima vez que se conecte deberá volverse a logear y volverá a haber constancia de sus datos.

Almacenamiento: Los mensajes que se guarden en la base de datos deben de estar cifrados, así como todas las claves que se necesiten almacenar.

Integridad

Transporte de red: Emplear, además de un protocolo seguro en el transporte de red, certificados para la integridad de los mensajes.

Disponibilidad

Tolerancia a fallos: El software debe ser tolerante a fallos sin dejar que el usuario introduzca valores no adecuados, y en caso de fallos siempre controlarlos e informar al usuario cuando proceda..

Para cuando el sistema se pusiera en producción se requerirían: servidores escalables y ante un problema con la base de datos optar por emplear la técnica fail open. No se dejaría a los usuarios sin servicio.

Requisitos de seguridad

En cuanto a los objetivos a lograr respecto a la seguridad del software contamos con los siguientes requisitos de seguridad a lograr en la aplicación:

- **Cifrado en tiempo real**: El chat será interactivo y sin latencias.
- **Mecanismo de autenticación seguro**: gestión de contraseñas utilizando SHA2 y Bcrypt.
- **Transporte de red seguro**: La comunicación entre el cliente y el servidor será vía TLS y la comunicación entre el cliente y el cliente para el navegador en Angular JS será vía HTTPS.
- **Optimización de la privacidad**: El servidor solo recibirá datos cifrados del cliente. No tendrá conocimiento en ningún momento sobre las claves de los usuarios o los mensajes de los mismos. Sus claves a guardar en la base de datos (su clave privada, la clave con la que se cifran los mensajes de un chat…) siempre llegarán cifradas al servidor. Los mensajes serán cifrados en el cliente, no en el momento de guardarlo en la base de datos desde el servidor, si no que el cliente los cifrará antes de comunicarlos al servidor.

Legislación

No es legal ofrecer los datos privados de un usuario sin conocimiento debido a la Ley Orgánica de protección de datos de carácter personal. En nuestro software no solicitamos dicho permiso y no publicaremos ningún dato de nuestros usuarios. Además, respecto a su información personal, no les pedimos datos demasiado privados, tan solo un nombre con el que deseen identificarse, una contraseña para el inicio de sesión y un estado que bien puede ser vacío o contener cualquier descripción que los usuarios deseen. Estos datos no serán compartidos con ninguna entidad o individuo, tampoco los mensajes que los usuarios mantienen entre ellos. El propio software hace que el servidor no conozca ningún dato legible sobre los mensajes o las contraseñas de los usuarios, es decir, ni quiera nuestro servidor al que podemos acceder o ver que está sucediendo, contiene estos datos en claro de los usuarios (mensajes y claves).

Privacidad

La privacidad consiste en permitir a los usuarios controlar el uso, recolección y distribución de su información personal y sus datos. En nuestro software el servidor no conoce ningún dato respecto a las claves o los mensajes del usuario, pues los recibe siempre cifrados. El servidor

tiene conocimiento cero, por lo que ni siquiera nosotros tenemos conocimiento de esta información del usuario.

Amenazas

A continuación hablaremos de las posibles amenazas hacia nuestro software, accidentales e intencionadas. Necesitamos conocer bien al enemigo, es decir, los posibles ataques a nuestro software o a sus usuarios.

Accidentales

Las amenazas accidentales provenientes de errores de los usuarios, catástrofes naturales, etc. Pueden ser los siguientes:

- **Introducción de datos**: El usuario introduce datos en un formato inadecuado (letras donde solo se piden números…).
- **Introducción de contraseñas débiles**: EL que las contraseñas sean demasiado cortas.
- **Destrucción de la BD**: Que el servidor donde se aloja la base de datos sea destruido o dañado por accidente (desastres medioambientales…), perdiendo así los datos de los usuarios, sus mensajes, etc.

Intencionadas

Los ataques intencionados pueden ser externos o internos.

Externas

- **Robo de BD**: El que un atacante robe la base de datos.
- **Intercepción MITM (man in the middle)**: El que un atacante intercepte la comunicación, obteniendo los mensajes y retransmitiéndolos a su destino sin cambiarlos (MTIM pasivo) o modificándolos (MTIM activo).
- **Intercepción reproducción**: Un atacante almacena la información que intercepta de las comunicaciones para utilizarlas posteriormente, valioso para obtener credenciales.
- **DDos**: Atacantes masivos a los servidores provocando que los mismos se saturen sin poder dar servicio a los verdaderos usuarios no maliciosos, o provocando problemas y daños en dichos servidores.
- **Inyecciones SQL**: intentos de un atacante de insertar comandos en nuestro servidor SQL. El atacante trata de introducir en campos de entrada de datos apóstrofes comprobando su efecto, puntos y coma para terminar dicha sentencia y ejecutar otra más como ver los datos de una base de datos, modificar alguna fila, eliminar tablas, etc.

- **Ataques al cliente o XSS**: Que un atacante vea que un servidor es vulnerable y inyecte contenido malicioso (javascript, flash…) y cuando la víctima visite la web dicho script se descargue desde su navegador ejecutándose y descargando malware del servidor instalándolo en el cliente.
- **Manipulación de cabeceras**: debido a que no utilizamos ningún valor proporcionado en las cabeceras de una petición HTTP, no tendríamos problemas en este sentido (inyecciones SQL en cabeceras como accept-language, o posibilidad de acceder a directorios restringidos en un servidor, etc).
- **Cookies**: Al no emplearse cookies no habría problemas de robo de cookies de los usuarios para hacerse pasar por ellos, o usar el monitoreo de la navegación del usuario para otros fines maliciosos.
- **Asalto de sesión**: un atacante roba el token que el servidor le proporciona a un cliente que inicia sesión, para identificarlo con dicho token.
- **Robo de identidad**: un atacante roba la identidad de un usuario mediante algún ataque como por ejemplo consiguiendo descifrar su contraseña por ser demasiado fácil, etc.

Internas

- **Empleado descontento**: un empleado instale malware en los servidores, robe la base de datos, etc.

Controles

A continuación hablaremos de los controles a llevar a cabo para evitar las amenazas que pueden poner en peligro al software o a los propios clientes y sus datos. Igual que necesitamos conocer los posibles ataques, necesitamos conocer nuestras posibles defensas y controles. Clasificaremos estos controles en técnicos, operacionales o de gestión y los agruparemos por familias de seguridad (identificación, control de acceso, auditoría…)
Algunos controles son proactivos evitando que ciertas amenazas se den (por ejemplo el dar seguridad a nivel de transporte con TLS, cifrar los mensajes y claves, etc). Otros son reactivos, estos sirven para reaccionar cuando un determinado ataque se dé (un ejemplo es, ante un ataque DDos, limitar la tasa de tráfico específico).

Técnicos

Identificación y autentificación

- **Sistema de autentificación basado en SHA2 y Bcrypt**: La autenticación es a través de conocimiento (contraseña). Cuando un usuario se registra con una contraseña, esta se hashea con SHA2. Parte de la contraseña sirve para cifrar claves de mensajes, otras claves... La otra parte sirve para el inicio de sesión, a la que se aplica Bcrypt guardando el resultado junto con la Salt generada de forma aleatoria. Cuando el usuario inicia sesión se comprueba que dicho dato generado con bcrypt coincida con volver a calcular Bcrypt de la mitad del hash (con SHA2) de la contraseña que introduzca el usuario, pero esta vez con la Salt guardada en la base de datos (generada en el registro). Este control nos ayuda a evitar los ataques al usuario a través del robo de su token de identificación, pues no necesitamos usarlo. Evitamos ataques como robos de identidad (no utilizamos tokens ni variables de sesión). También se evitan ataques de intercepción (reproducción) ya que, aunque el cliente le pase datos al servidor, respecto a la contraseña, solo le pasa la mitad del hash que se ha creado con SHA2 de la contraseña del cliente. El atacante no podrá averiguar la contraseña a partir de dicho dato, no le servirá para robarle la identidad. Aunque algún atacante robara la base de datos no le valdría con saber la mitad del hash con SHA2 y cifrada con Bcrypt de la contraseña del cliente y el valor Salt.
- **Mapa de conexiones**: En este map las claves son los id de los usuarios y los valores son las conexiones de estos clientes. Cuando un cliente se pone en marcha el servidor detecta dicha conexión, solo la añadirá al map de conexiones cuando el usuario inicie

sesión (o tras registrarse, pues ya iniciaría sesión). Se añadiría el id de dicho usuario como clave, y como valor la conexión establecida. A partir de este momento el cliente ya se puede comunicar con el servidor y viceversa. Cuando el servidor obtiene una petición por parte de un cliente, comprueba al usuario asociado al socket, es decir a la conexión. Con este control conseguimos una buena autentificación, evitando que un atacante se haga pasar por un usuario.

Control de acceso

Todos los usuarios del software tienen el mismo nivel de autorización en el software.

Disponibilidad

- **Tolerancia a fallos**: se controla evitando que el usuario pueda realizar acciones o introducir datos que provoquen fallos en el sistema. Si por ejemplo edita sus datos dejando el campo de su nombre vacío, esta acción no se llevará a cabo. Lo mismo ocurre con el nombre a editar de un chat (cuando un usuario crea un chat este se crea con un nombre por defecto que luego se puede editar). Cuando un usuario agrega o elimina a un usuario de un chat, lo hace a través de un select que le muestra los usuarios que hay en la BD. De esta forma, no puede introducir datos incorrectos. Además tampoco dejamos que un usuario pueda enviar mensajes vacíos.
- **Limitar tasa de tráfico específico**: Para evitar que un ataque vía DDos se dé provocando que el servidor no responda correctamente a los usuarios que utilicen el software, cuando la aplicación se ponga en producción se llevarán a cabo medidas concentradas en limitar la cantidad de tráfico específico, por ejemplo, que solo los usuarios que hayan iniciado sesión puedan utilizar el sistema, a los demás no se les escuchará sus peticiones.
- **Servidores escalables**: Con vistas a soportar un ataque DDos tendríamos nuestros servicios alojados en servidores escalables que vayan aumentando a medida que aumenta la cantidad de usuarios conectados.
- **Distribución de carga**: En el caso de tener ataques de que el servidor donde se aloja la BD sufra algún accidente o daño, en el momento de poner en producción el software la base de datos soportaría distribución de carga y replicación de datos. Cabe señalar que también buscaríamos una base de datos que creciera automáticamente según hayan más usuarios solicitando los servicios del software. Esto nos ayudaría también a controlar una posible destrucción del servidor de la base de datos.

- **Estrategia fail open**: Al tener el sistema en producción, en caso de fallo dejaríamos el sistema abierto (fail open) intentando solucionarlo, pero sin dejar sin servicio a los usuarios.

Confidencialidad

- **Claves del usuario cifradas**: Las claves de los usuarios están protegidas empleando una combinación de SHA2 y Bcrypt. En la base de datos se guarda la parte de la contraseña hasheada (para el login) con bcrypt. Se evita el precálculo o ataques por GPU a las contraseñas de los usuarios. Las claves de los usuarios para cifrar los distintos mensajes se generan aleatoriamente y se cifran con AES. Ni siquiera el servidor tiene conocimiento de la contraseña del usuario ni de estas claves, y por supuesto tampoco de los mensajes al estar cifrados con estas claves del usuario también cifradas. Aunque un atacante robara la BD o consiguiera obtener datos en las comunicaciones, no obtendría datos legibles o valiosos para suplantar a un usuario, averiguar claves, etc.

  Nota: La misma clave se guarda varias veces, para cada usuario de una forma distinta, ya que se guarda la clave pero cifrada con AES teniendo como clave, la mitad del hash de la clave del login de dicho usuario.
- **Cambio de claves**: Cuando un usuario es eliminado de un chat, se cambia la clave con la que se cifran los mensaje. De esta forma el usuario ya no tiene acceso a dichos mensajes, y aunque fuera un usuario experto y accediera a dichos datos o los interceptará, no podría descifrarlos al no tener la clave.
- **Para los mensajes de los chats**: los mensajes se cifran y así es como se guardan en la base de datos. Aún en caso de robo de los datos por intercepción en una comunicación cliente-servidor o robo de la base de datos, la información sería ininteligible. Se garantiza que un usuario que ya no esté en la conversación no pueda entender dichos mensajes aunque los consiguiera obtener, ya que al eliminarse un usuario, la clave para cifrar los mensajes de dicho chat, se vuelve a generar y no se guarda para dicho usuario que ha sido eliminado de la conversación. Nos protegemos de un ataque MITM, aunque obtenga los datos, no los podrá entender, es decir, descifrar.
- **Transporte de red**: la comunicación entre el servidor y el cliente en Go viajan por TLS, el cual nos ofrece seguridad a nivel de transporte. Así mismo la comunicación entre el cliente en Go y el cliente web (Angular JS) es por HTTPS, es decir, HTTP sobre SSL/TLS. Como vemos, en todo momento se cuida que las comunicaciones a nivel de

transporte de red estén protegidas con protocolos conocidos y seguros. Estos controles nos protegen frente a ataques como MITM.

- **Protección frente a inyecciones SQL**: Cuando se envían mensajes en los diferentes chats, no se puede producir una inyección SQL debido a que este valor se cifra y así es como se guarda. No es posible que la consulta a la BD vea una sentencia que intenta ejecutar una inyección SQL, como ya hemos dicho nuestro servidor tiene un conocimiento cero respecto a esto. En otros casos como añadir/eliminar usuarios de un chat, editar la información de un usuario o un chat… las inyecciones SQL no son posibles, ya que las librerías (gorp) que utilizamos para comunicarnos con nuestra base de datos en MySQL no lo hacen viable. Algunos ejemplos de intentos de ataque serían:
    - Si el usuario intentara iniciar sesión introduciendo por ejemplo un nombre igual a "*Pepe*" y una contraseña igual a "*' OR ''='*" como realizamos una comprobación con Bcrypt, este dato no llegaría a la consulta a la base de datos, dando un error de inicio de sesión.
    - Si un usuario introdujera un mensaje como "*Alicia'; SELECT * FROM usuario WHERE nombre LIKE '%*" MySQL no ejecutaría esta sentencia de mostrar la tabla usuarios.

Integridad

- **Protección frente a inyecciones SQL:** Esta protección ya comentada también nos ayuda frente a la modificación o eliminación de datos en la base de datos.
- **Transporte de red seguro y certificados**: En las distintas comunicaciones se emplea TLS con certificados, por lo que nos protegemos de atacantes que traten de modificar el mensaje, es decir, se asegura la integridad de los mensajes.

Auditoría

- **Controles a los servidores**: Al poner el sistema en producción, se realizarán controles a los servidores donde esté alojado el programa, buscando malware en los mismos evitando así ataques al cliente o XSS. También se puede optar por confiar el despliegue a servidores con una reputación conocida de seguridad en sus servicios.
- **Revisión de código**: También se revisaría el código fuente de forma periódica (cada x meses), en busca de vulnerabilidades.

Operacionales

Alerta

- Los métodos de alerta ante un incidente serán los más inmediatos, teléfono por ejemplo.

Respuesta a incidentes

- Los desarrolladores que han participado en este proyecto serán alertados en caso de incidentes, para que inmediatamente los que estén disponibles puedan solventar los incidentes.

Monitorización

- Al poner en producción el software, se monitorizaría el software para actuar rápido frente a posibles amenazas o errores. También se monitorizarían las actividades de administración y funcionalidad de forma que se pueda establecer un historial de transacciones en caso de necesitarlo.

Protección de medios

- **Acceso a servidores restringido**: Para evitar que un empleado descontento pueda introducir código malicioso en el servidor o robe la base de datos, estas tareas las realizarán los dueños del software y solo tendrán acceso al servidor dichos dueños.

Gestión

Certificación

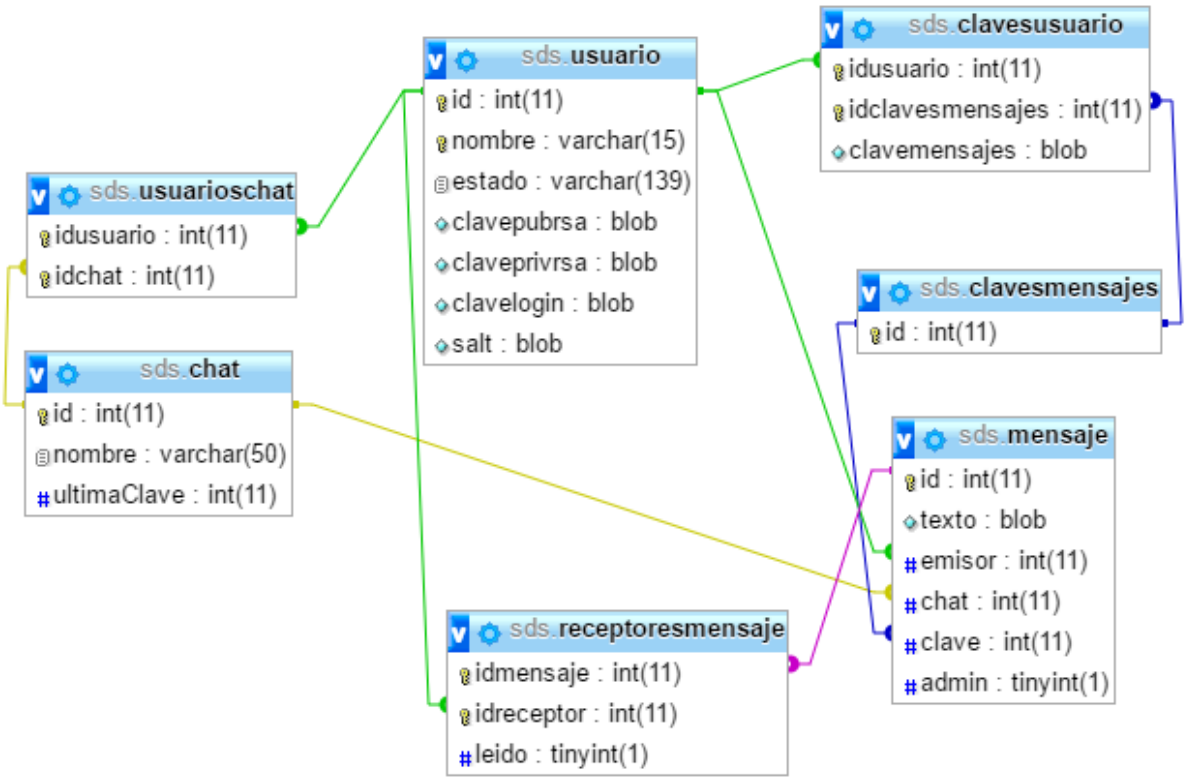
- **Certificados**: Se emplean certificados en todos los servidores. Tanto entre el servidor que se encarga de realizar distintas operaciones en la base de datos (obtener y devolver datos, modificar datos, insertarlos o borrarlos) y el cliente en go, como entre el cliente en .go con el cliente web (Angular JS). De esta forma, evitamos que un atacante se haga pasar por el servidor engañado al cliente en .go, o hacerse pasar por el cliente en .go engañando al cliente web.

Información del plan

La fecha final para aprobar y finalizar este documento que describe el plan de seguridad es el día 06/06/2016.
En los siguientes apartados especificaremos más en concreto nuestro plan de seguridad.

Descripción detallada del software

Base de datos



Usuario: Los usuarios tienen un código de identificación id que no se puede repetir. El nombre del usuario tampoco puede repetirse, siendo único. Cada usuario tendrá su contraseña a la que se aplicará un hash. Este valor se dividirá en dos, la primera parte servirá para el login y la segunda parte servirá para cifrar las claves con las que descifrar sus mensajes. La parte para el login se hashea con bcrypt guardándose en la columna clavelogin y con la Salt generada necesaria para bcrypt guardada en la columna Salt. Cada usuario tendrá una clave pública y una clave privada de RSA (la clave privada se guarda cifrada en el servidor con la clave de login del usuario). También contiene un atributo estado que aparece en su perfil.

Chat: Un chat contiene su identificador y un nombre. Además contiene el id de la clave con la que actualmente se están cifrando los mensajes de dicho chat.

UsuariosChat: En esta tabla se relaciona el id de un chat, con el id de cada uno de los usuarios que se pertenecen al chat.

Mensaje: Se corresponde a los mensajes que se envían en los diferentes chats. Cada mensaje tiene un texto (cifrado), el id de un emisor (un usuario), el id de un chat y el id de la clave con la

que se ha cifrado. También contiene un valor booleano denominado admin, el cual nos informa de si es un mensaje de administrador o no.

ReceptoresMensajes: Esta tabla contiene el id de un mensaje, el id de un usuario que es receptor de dicho mensaje y un valor booleano llamado leído. Esta tabla sirve para saber que usuarios de un chat (siempre que no sean ellos los emisores del mensaje) han leído un mensaje.

ClavesMensajes: En esta tabla se guardará el id de la clave con la que se cifra uno o varios mensajes.

ClavesUsuario: Cada usuario guardará en esta tabla su identificación como usuario, la identificación de la clave que quiere guardar, y el dato de dicha clave pero cifrada con su clave (la segunda parte del hash realizado a su contraseña en claro). Por ejemplo, si un mensaje se cifra con el id de clave 1, cada usuario perteneciente al chat de ese mensaje, tendrá guardada la clave asociada a dicho id pero cifrada con su clave de login.

Archivos

Servidor Go

BD (go): Este archivo contiene la estructura de los datos de la BD a la que nos conectamos con MySQL. También un método que realiza lo necesario para conectarse con la base de datos utilizando la librería gorp (señala las tablas que hay y con qué estructura, es decir, struct coinciden). Devuelve un DbMap con el que poder realizar consultas a la BD.

UsuarioBD (go): Contiene las estructuras necesarias para manejar datos de usuarios. Se encarga de todas las operaciones a realizar en la tabla usuario (insertar un nuevo usuario, editarlo, devolver sus datos, devolver la clave pública de un usuario, devolver todos los usuarios que hay en la BD, etc.)

MensajeBD (go): Contiene las estructuras necesarias para manejar datos de mensajes. Se encarga de todas las operaciones a realizar respecto de los mensajes de un chat. Por ejemplo: insertar un mensaje en un chat, devolver todos los mensajes de un chat, obtener la clave con la que se cifra un mensaje, insertar una nueva clave para los mensajes, marcar un mensaje como leído, etc.

ChatBD (go): Contiene las estructuras necesarias para manejar datos de los chats. Se encarga de realizar consultas en la BD relacionadas con los chats, por ejemplo: obtener todos los chats de un uuario, crear un chat, modificar chat, asociar la clave actual con la que se están cifrando los mensajes del chat, añadir usuarios a un chat, eliminar usuarios de un chat, comprobar si un usuario está en un chat, marcar los mensajes de un chat entero como leído, etc.

Constants (go): Contiene constantes numéricas para que en el router se identifique que tipo de solicitud está realizando un cliente y contestarle con un código de operación correcta o fallida.

Server (go): Este archivo realiza la principal función como servidor. Por cada cliente que establece conexión con el cliente se crea un hilo de ejecución. Cada vez que le llegue una petición de un cliente, llamará a una función del archivo router que se encarga de procesar dicha petición.

Router (go): Contiene la estructura con la que se comunican el lado del servidor y el lado del cliente. Tiene una única y gran función que se encarga de manejar todas las peticiones del cliente. Según la petición que sea (la constante con valor numérico del archivo constants) realiza una acción u otra. Suelen ser llamadas a los archivos que realizan operaciones en la base de datos. Si la operación se realiza correctamente se envía una constante que lo indica y lo mismo si la operación no se realiza correctamente.Junto a esta información a veces devuelve datos de la BD, mensajes para el usuario, claves, etc.

Cliente Go

Constants (go): Contiene constantes numéricas para que al realizar peticiones al servidor el cliente le especifique que tipo de solicitud está realizando. Ademas cuando reciba la contestación del servidor sabrá por la constante, si la solicitud se ha realizado correctamente o no. También contiene las estructuras necesarias para manejar datos como los usuarios, mensajes de un chat y toda su información, chats, todos los datos de un chat y los mensajes del tipo administrador. También la estructura utilizada para comunicarse con el servidor.

Auxiliares (go): Contiene funciones auxiliares como escribir en un socket para comunicarse con el servidor o con el cliente web, generar claves RSA, cifrar y descifrar con RSA y también con AES, etc.

Client (go): Se encarga de realizar peticiones al servidor. Estas peticiones son las que luego meneja el archivo router de la parte del servidor.

ClienteHttp (go): Se encarga de mostrar el archivo HTML junto con el archivo JS que contiene el código de Angular JS. Este archivo hace de mediador entre el cliente web y el archivo client. Cuando el cliente web realiza solicitudes, este archivo se encarga de procesarlas y realizar la correspondiente llamada o llamadas a funciones del archivo client.

Cliente Web Angular JS

Index (html): Se encuentra el contenido HTML que muestra la interfaz de la aplicación. Los datos dinámicos a mostrar se manejan con Angular JS. También contiene una serie de estilos en CSS3, para darle formato de colores, espaciados… a la interfaz.

Index (JS): Este archivo contiene al controlador realizado en Angular JS. Se encarga de rellenar los datos que el HTML debe mostrar, definir las funciones a realizar cuando se clica un elementos, etc. Realiza peticiones que se escriben en un socket para que el archivo client.go las procese. También procesa los datos que el archivo client.go le envía o reenvía de parte del servidor.

Ejecución

Servidor

Para ejecutar el servidor debemos de situarnos por consola en el directorio “servidor” del proyecto. Una vez dentro podemos crear un ejecutable con la siguiente instrucción:

go build

Esto nos creará el ejecutable servidor.exe, el cual podremos ejecutar.

Otra opción es ejecutar simplemente:

go run server.go router.go mensajeBD.go usuarioBD.go chatBD.go BD.go constants.go

Cliente

Para ejecutar el cliente debemos de situarnos por consola en el directorio “cliente” del proyecto. Una vez dentro podemos crear un ejecutable con la siguiente instrucción:

go build

Esto nos creará el ejecutable cliente.exe, el cual podremos ejecutar.

Otra opción es ejecutar simplemente:

go run client.go clientehttp.go constants.go auxiliares.go

Al ejecutar el cliente nos pedirá que escribamos el puerto por donde se creará la conexión entre el cliente y cliente web.

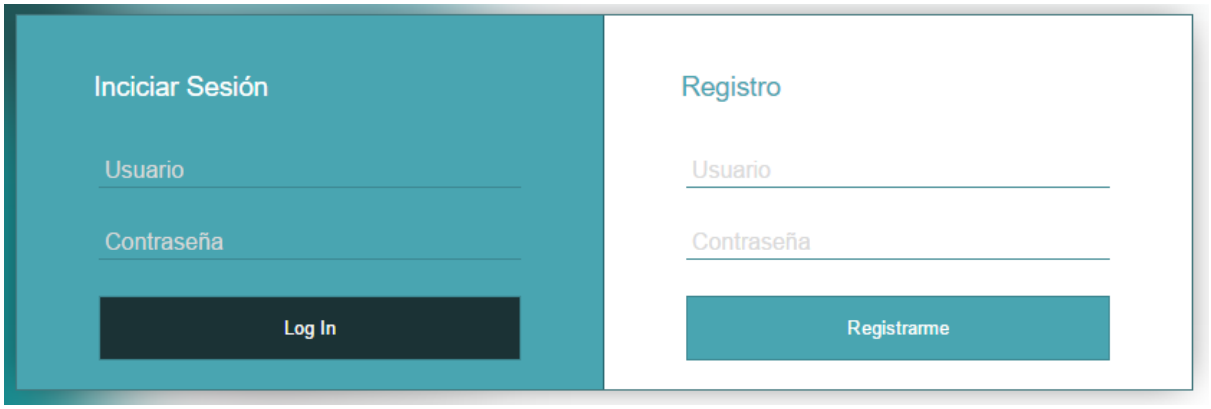
Cliente web

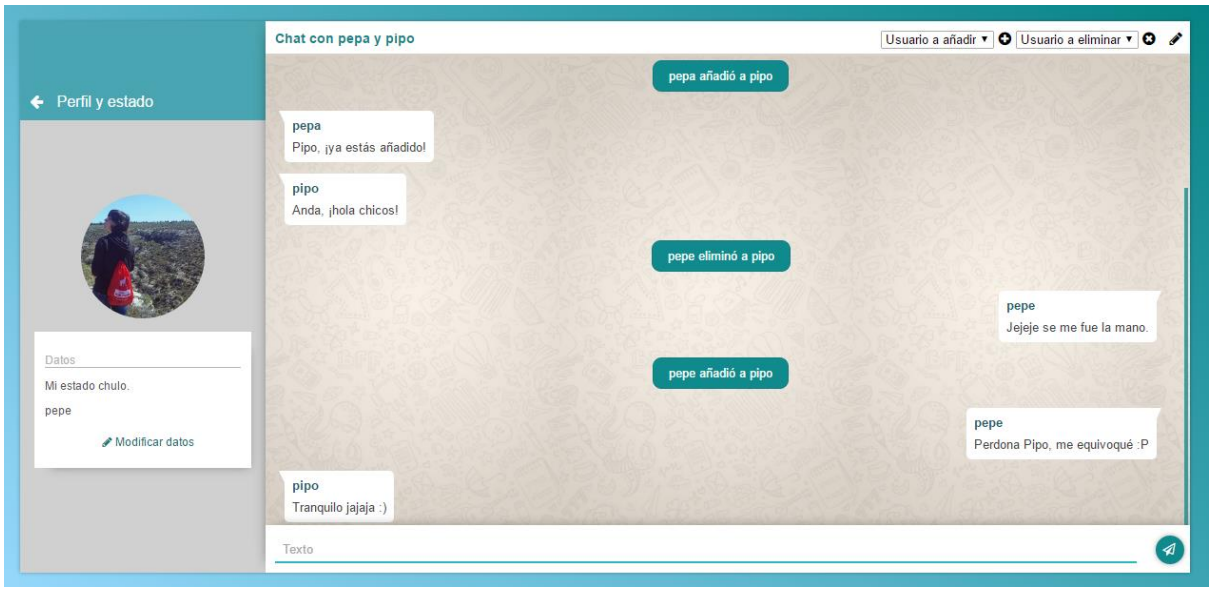
Podemos abrir la interfaz web con el servidor y el cliente desplegados abriendo un navegador y yendo a la dirección: https://localhost:<puerto elegido>.

Nota: Se deberán instalar con la instrucción “go get” los paquetes importados en el proyecto que no vengan por defecto con la instalación de go.

Imágenes interfaz

A continuación mostramos unos ejemplos de la interfaz que se muestra con el cliente web en Chrome.

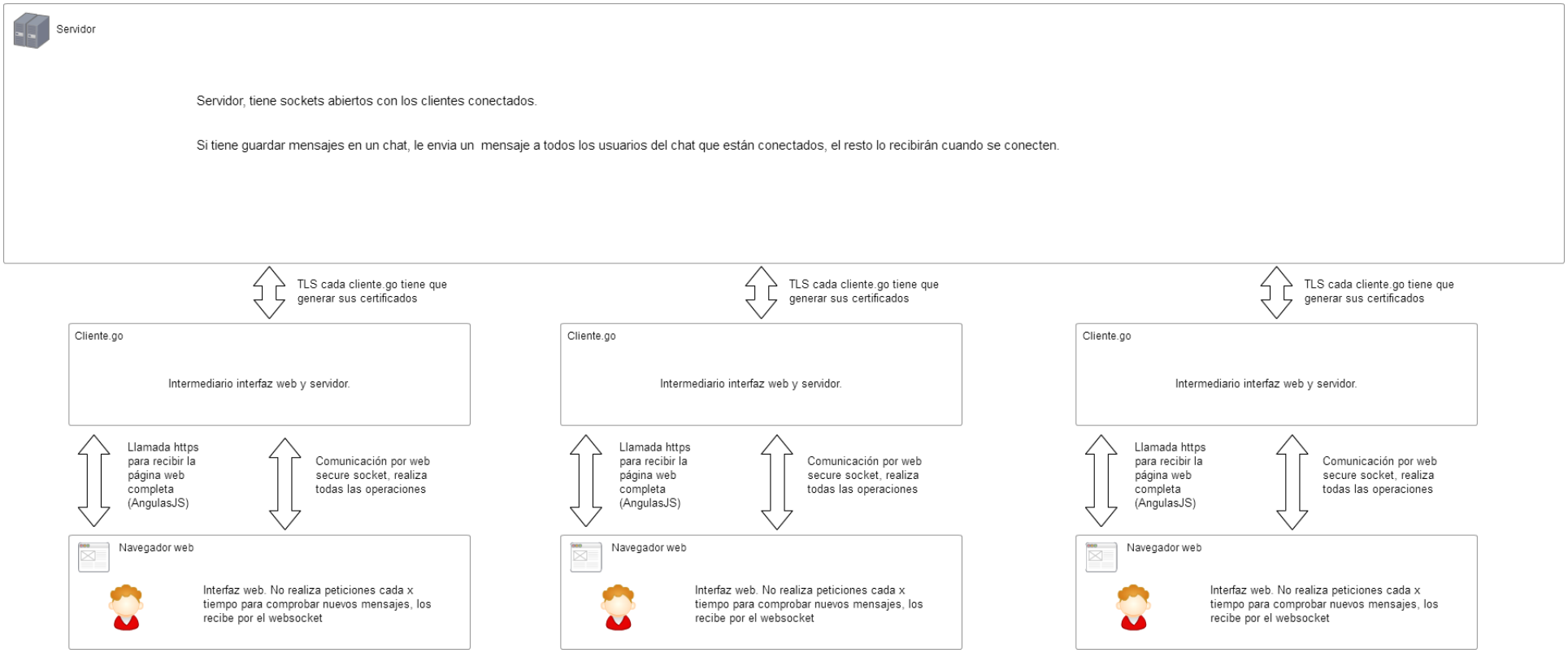




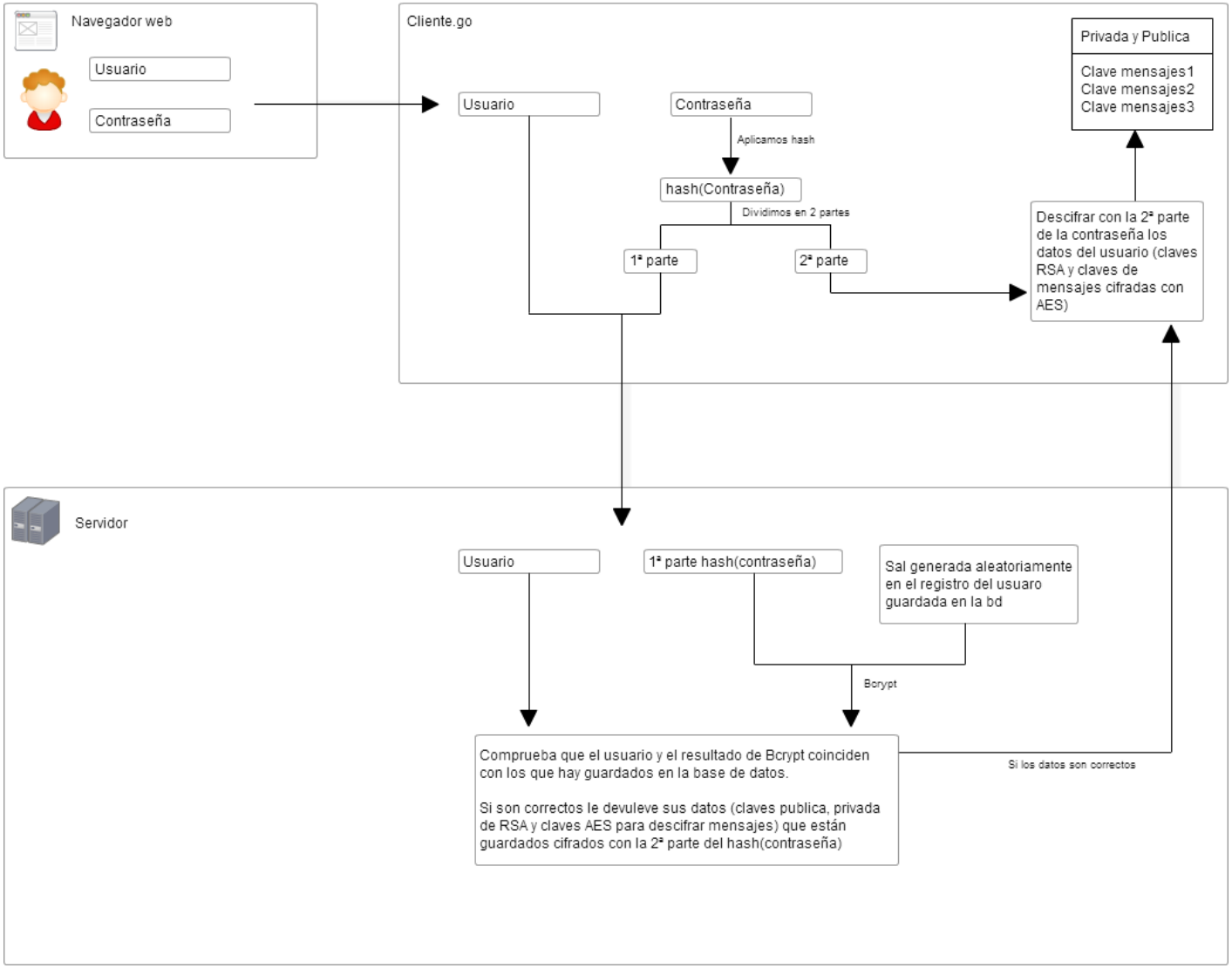
Esquemas del funcionamiento

A continuación mostraremos una serie de esquemas que son ejemplos de ejecuciones del programa.

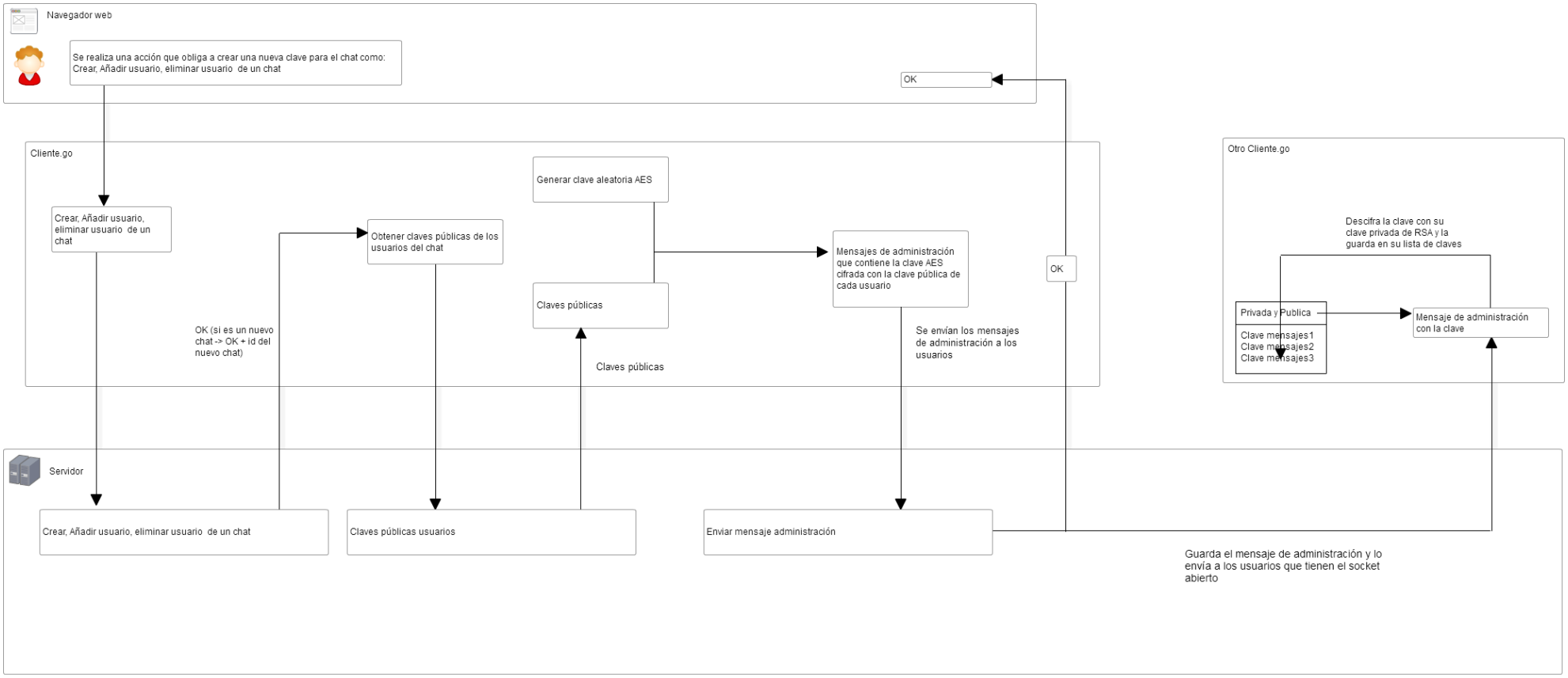
En el siguiente esquema podemos observar como es la conexión entre el servidor y el cliente, además de la conexión entre el cliente y cliente web.



En este esquema podemos ver el proceso de inicio de sesión de un usuario.



En el siguiente esquema podemos ver como se generan las claves a la hora de crear un chat, eliminar un usuario o añadir un usuario a un chat.



Este esquema está simplificado, hay llamadas como guardar la nueva clave cifrándola con la segunda parte del hash de su clave en el servidor o realizar una petición para obtener un nuevo id para la clave nueva que no se muestran

En este esquema vemos cómo es el proceso de enviar un mensaje.