UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Lector de manga distribuido para escritorio (Windows, Linux, MacOS)

Proyecto Final de Carrera

Ingeniería Técnica Superior de Informática

Autor: Adrián García Santiago

Director: Juan Luís Posadas Yagüe

Curso académico 2014-2015

Agradecimientos

Este proyecto no habría sido posible sin la ayuda y apoyo de varias personas que de una u otra forma han ayudado a la realización del mismo.

Primero agradecer al director del proyecto, el Dr. Juan Luís Posadas Yagüe, por su interés con el proyecto y la ayuda prestada.

A mi compañero Juan Manuel Fernández Nácher por la implementación del cliente para Android y por su ayuda con el diseño e implementación del servidor.

Y por último agradecer a familiares y amigos por su ayuda y apoyo prestado.

Resumen

En este proyecto se ha implementado un lector de mangas, que es el nombre que reciben los cómics japoneses, que permite ser utilizado desde diversas plataformas (ordenadores y dispositivos móviles), manteniendo la información del usuario sincronizada entre todas ellas.

Se ha implementado tanto la versión de escritorio como el servidor. Éstos permiten mantener una lista de mangas favoritos y llevar la cuenta de capítulos leídos de cada manga teniendo esta información sincronizada entre los distintos dispositivos donde se instale la aplicación.

El cliente de escritorio se ha implementado en C++11. El desarrollo de la interfaz gráfica se apoya en el framework Qt, el cual además provee una serie librerías básicas, todas ellas multiplataforma, las cuales facilitan el desarrollo sin tener que depender de otras librerías externas.

El servidor se ha desarrollado con el lenguaje de programación Ruby. Éste, por una parte, busca la información de mangas en diversas webs dedicadas a ello, guardando esta información en una base de datos MongoDB. Y por otra parte, mediante el framework Sinatra ofrece una API REST para que las aplicaciones cliente se comuniquen con él obteniendo la información de los mangas y pudiendo guardar o leer el estado del usuario.

Palabras clave: Lector, manga, escritorio, C++, Qt, Ruby, MonoDB, Sinatra, REST.

Tabla de contenidos

Índice de ilustraciones

Índice de tablas

Índice de código

1. Introducción

1.1. Presentación del tema

Hay muchos tipos de cómics. Entre ellos está el cómic japonés, al cual se le llama “manga”.

En la mayoría de países el catálogo de mangas es bastante escaso. Pero hay una gran comunidad alrededor del manga, con aficionados por todas partes que se dedican a traducir estos cómics a su propio idioma en cuanto salen para que el resto de aficionados puedan disfrutarlo también. Ésta es la única manera de poder leer muchas obras que ninguna editorial ha querido importar a otros países.

1.2. Motivación

Existen páginas que recopilan estos mangas traducidos por aficionados, permitiendo la lectura online y todo organizado en una jerarquía de obras y capítulos. Pero estas webs no permiten llevar la cuenta de tus capítulos leídos, lo cual obliga a llevar la cuenta manualmente de alguna manera, siendo fácil olvidarse de marcar los capítulos, y si lees varios mangas a la vez teniendo que comparar manualmente con tu lista siempre que abres la web para ver en cuál ha salido algún capítulo nuevo. Además, la mayoría no ofrecen una experiencia de lectura óptima debido a las limitaciones de los navegadores web, siendo bastante molesto tener que esperar a que descargue la imagen cada vez que pasas de página.

1.3. Objetivos

Se persigue desarrollar una aplicación que facilite el acceso a estos mangas traducidos por aficionados que no tenga las limitaciones de estas webs.

Como base utilizará la información que extraerá de estas webs, pero permitirá al usuario llevar la cuenta de capítulos leídos de los mangas, permitiendo leer cada uno de una web distinta. Esto será automático, así que a medida que vaya leyendo el programa irá marcando los capítulos como leídos, aunque también permitirá al usuario marcar o desmarcar capítulos como leídos. Permitirá también seleccionar una serie de mangas favoritos, pudiendo ser cada uno de una web distinta, los cuales los mostrará en la primera pantalla de la aplicación, permitiendo ver de un vistazo si alguno tiene capítulos nuevos que no hayas leído. Toda esta información podrá sincronizarse entre distintas instancias del programa

que tengas instalado en diversos dispositivos. Por último, ofrece una experiencia de lectura a pantalla completa, eliminando distracciones y ofreciendo más superficie de lectura, en la cuál se descargan todas las imágenes rápidamente nada más abrir el capítulo, evitando descargas entre página y página.

2. Entorno

2.1. Introducción

En este apartado se explican los aspectos a tener en cuenta para desarrollar aplicaciones Qt para el cliente y servicios REST en Ruby para el servidor.

Además se hace un análisis de aplicaciones similares y las diferencias con la aplicación a desarrollar.

Tras esto se extraen una serie de conclusiones a partir de todo lo visto en este apartado.

2.2. Entorno del proyecto

Existen varios sistemas operativos de escritorio, todos ellos muy diferentes. Se busca en este proyecto realizar una aplicación que pueda funcionar en todos ellos proporcionando siempre un aspecto nativo a la aplicación en la mayoría de los casos para que quede bien integrada con el resto de aplicaciones del sistema.

El framework Qt nos proporciona todas estas características. Es capaz de funcionar en los principales sistemas operativos de escritorio: Linux, Mac y Windows. Es capaz de adaptar el estilo de su interfaz gráfica para quedar integrado en la mayoría de los sistemas e incluso en la mayoría de entornos de escritorio de Linux. Además, gracias a la gran cantidad de librerías base que proporciona, las cuales abstraen los componentes del sistema, es posible compartir todo o casi todo el código entre las compilaciones para los diferentes sistemas, siendo raro que se deba hacer una porción de código sólo para proporcionar una funcionalidad a un sistema concreto.

En cuanto al servidor se ha buscado una tecnología que funcione en servidores Linux, que sea moderna, simple, potente y escalable. Ruby y MongoDB proporcionan estas características.

2.3. Aplicaciones similares

2.3.1. CDisplay

Esta aplicación permite leer mangas que ya tengas descargados previamente. Mediante un navegador de archivos permite seleccionar el archivo a mostrar.

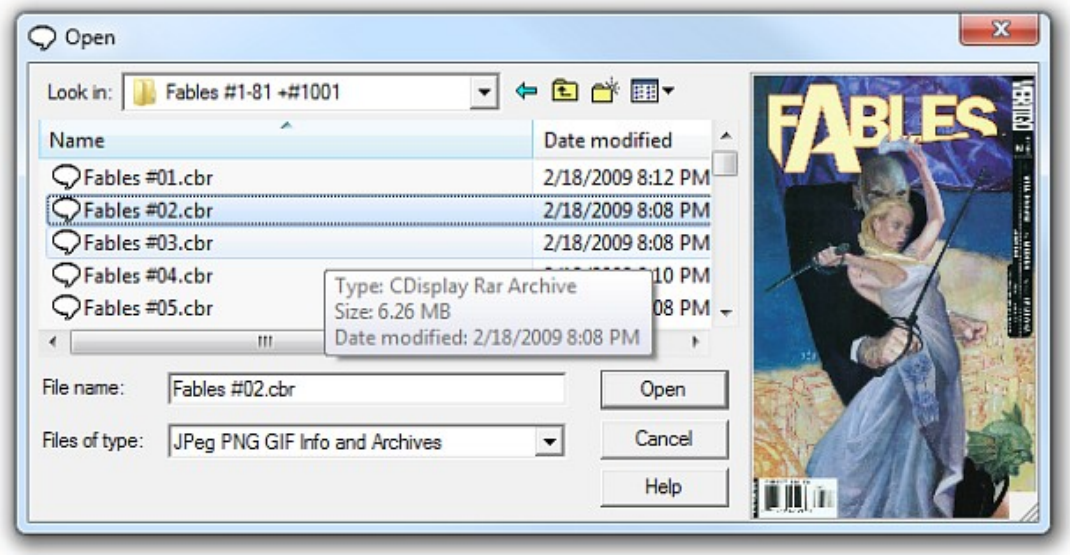

Ilustración 1: CDisplay 1 (Selector de archivos)

Presenta una interfaz de lectura simple, pero con todo lo importante.

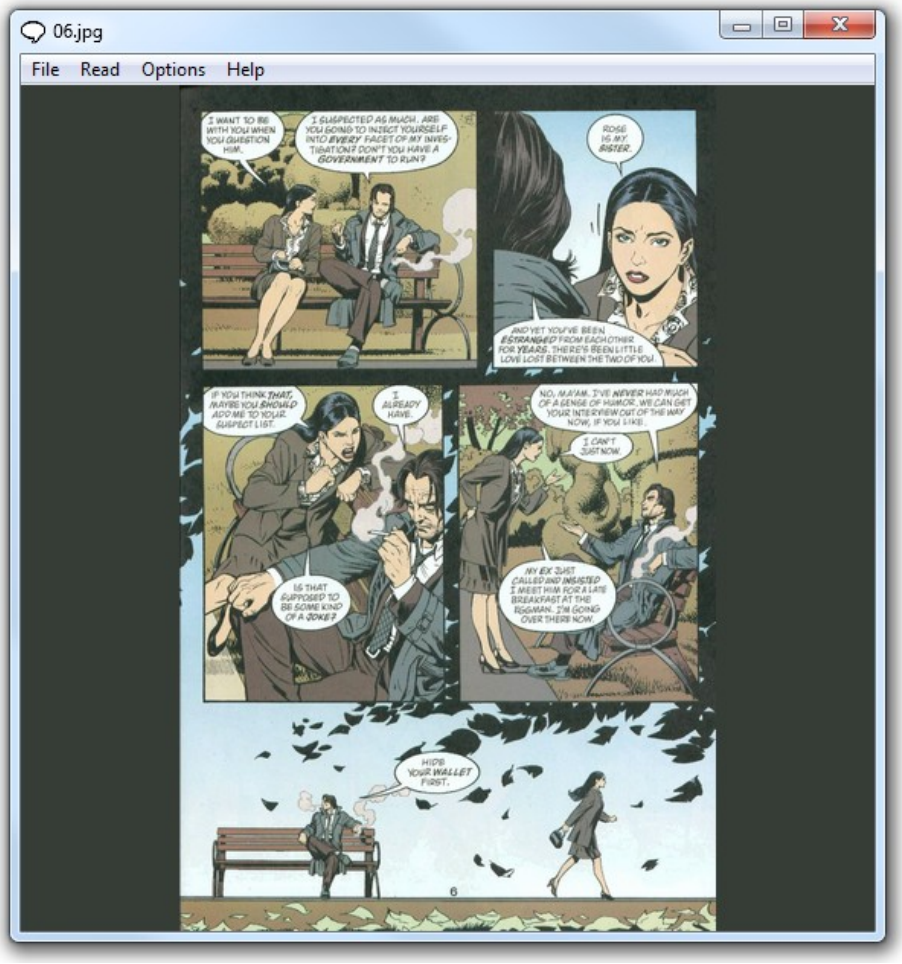

Ilustración 2: CDisplay 2 (Visor)

El visor tiene muchas opciones relativas a cómo mostrar las imágenes.

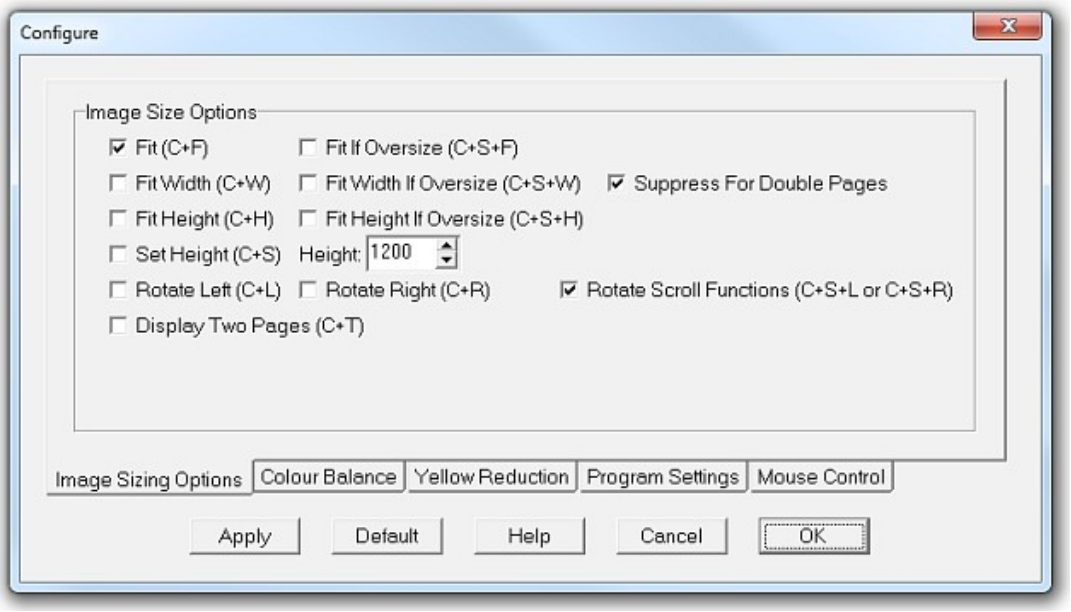

Ilustración 3: CDisaplay 3 (Cofiguración)

El programa no es capaz de buscar o descargar mangas por si sólo. Tampoco es capaz de marcar archivos como leídos. Todo esto debe hacerlo el usuario por si sólo.

2.3.2. GonVisor

Al igual que Cdisplay sólo es capaz de mostrar mangas que hayas descargado previamente por tu cuenta, y tiene las mismas limitaciones que la aterior aplicación.

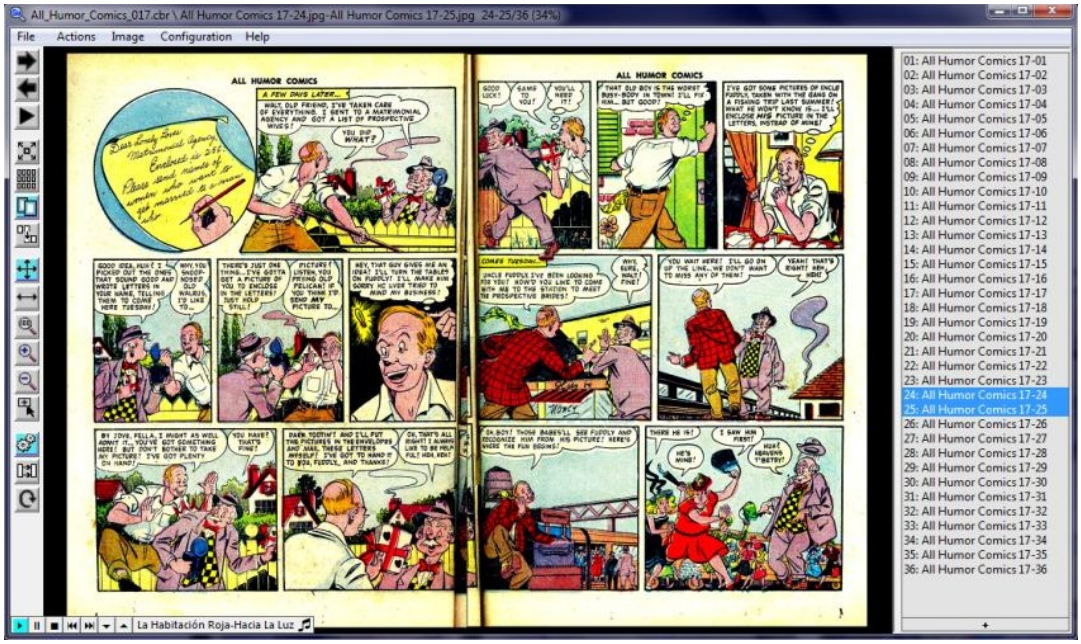

Ilustración 4: GonVisor 1 (Ventana de lectura)

Pero presenta algunas opciones interesantes, como una opción de auto-ajuste de imagen que puede lograr mejorar mucho la calidad de las páginas.

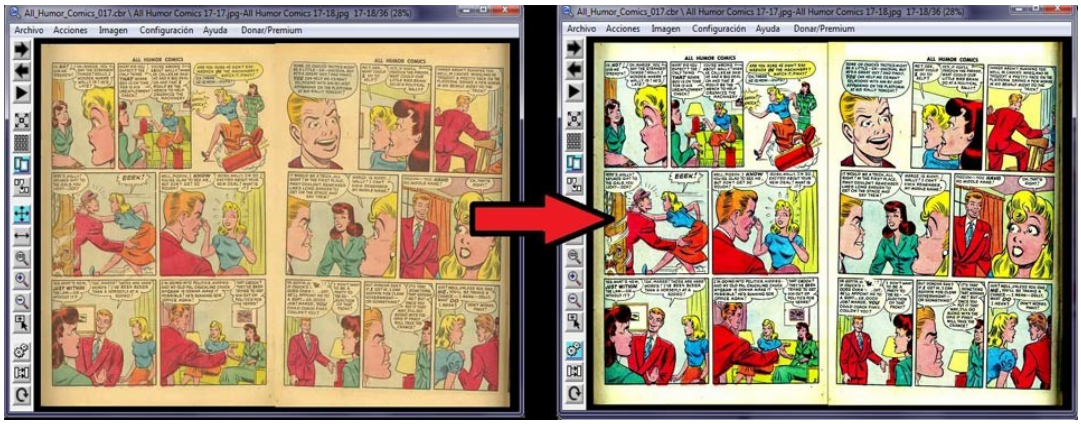

Ilustración 5: GonVisor 2 (Auto-ajuste de imagen)

También añade algunas opciones más como botones en la ventana por defecto haciéndolas más accesibles, sin tener que recordar la combinación de teclas que tiene asignada la opción.

2.3.3. ComicRack

Esta aplicación tiene clientes para Windows, Android e iPad. Tiene muchas opciones, pero las presenta en la interfaz de forma confusa, especialmente en la versión para Windows.

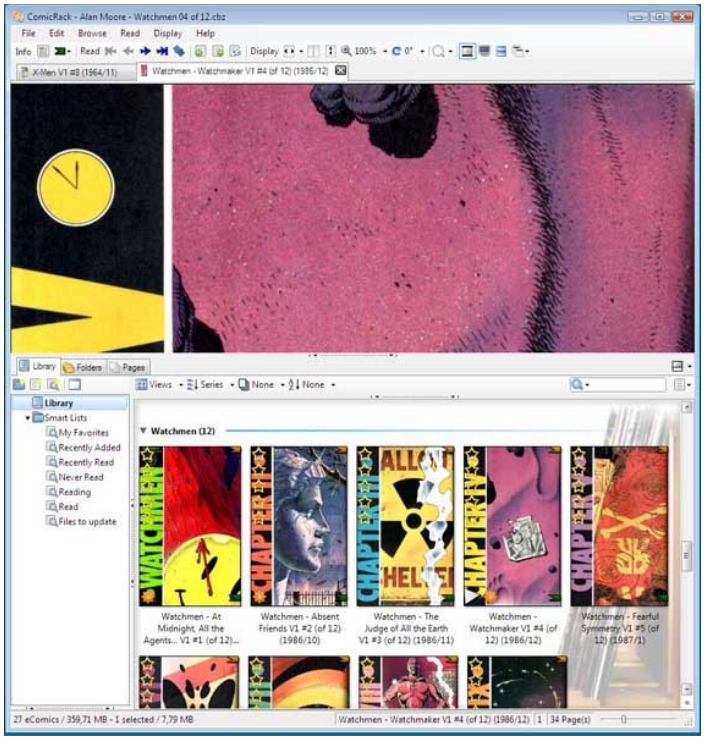

Ilustración 6: ComicRack 1 (Cliente de Windows)

No es capaz de buscar por si mismo mangas, sino que es el usuario el encargado de buscar y descargar los archivos, añadiéndolos al programa para que este los indexe en su librería y sea capaz de abrirlos. Pero a diferencia de los anteriores es capaz de recordar cuáles capítulos han sido ya leídos.

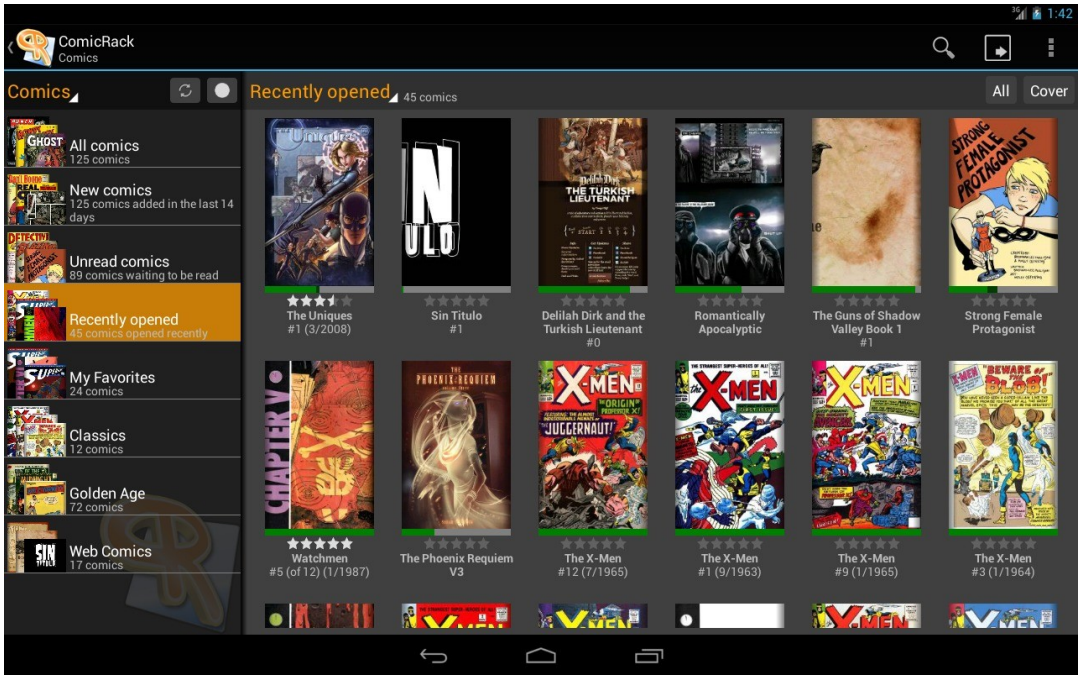

Ilustración 7: ComicRack 2 (Cliente de Android)

Puede sincronizar la librería que tiene en el cliente de Windows con los clientes móviles, incluyendo la cuenta de capítulos leídos. Pero para hacer esta sincronización requiere que ambos dispositivos estén encendidos y conectados a la misma red Wifi. No se guarda en la nube, pudiendo perder toda la información almacenada en el programa si borraras el cliente de Windows.

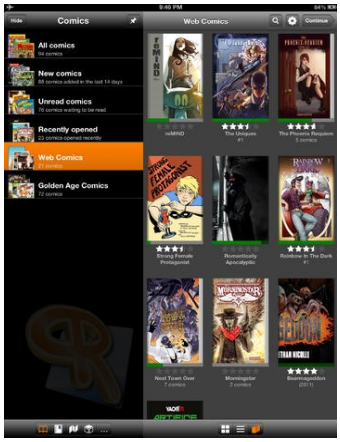

Ilustración 8: ComicRack 3 (Cliente de iPad)

2.3.4. HakuNeko

Este progama tiene versión tanto para Windows como para Linux.

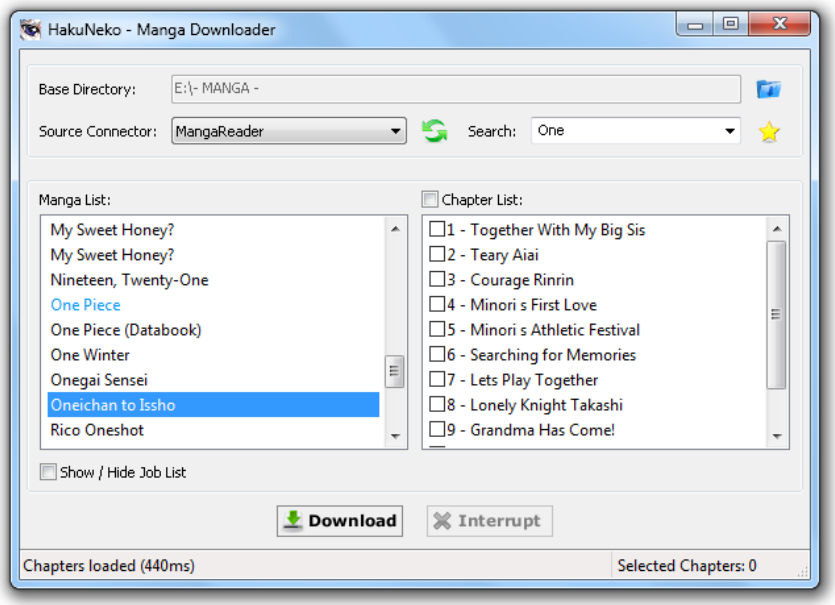

Ilustración 9: HakuNeko 1 (Versión para Windows)

El programa escanea las páginas de mangas presentando una lista de mangas y sus capítulos en cada web. Permite seleccionar capítulos y añadirlos a una lista de descarga. Luego procesa esta lista descargando cada capítulo y creando un archivo con el capítulo para poder leerlo posteriormente mediante otro programa.

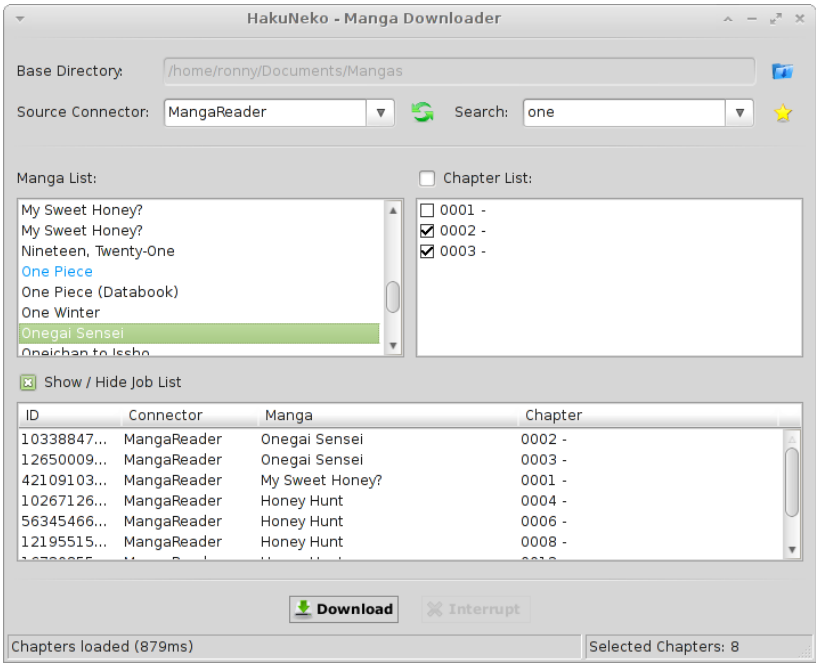

Ilustración 10: HakuNeko 2 (Versión para Linux)

Este programa tiene el problema de que no tiene un visor para leer los capítulos que descarga, no teniendo tampoco ningún tipo de gestor de librería que organice los elementos ya descargados o que lleve la cuenta de capítulos leídos. Tampoco es capaz de recordar los capítulos que ya ha descargado.

2.3.5. Comic Shelf

Este programa, el cual sólo está disponible para Windows, es hasta ahora el más completo si quisiéramos utilizarlo únicamente desde un dispositivo el cual ejecute el Sistema Operativo Windows. Aun así presenta algunas carencias a parte de la falta de cualquier tipo de sincronización con otros clientes.

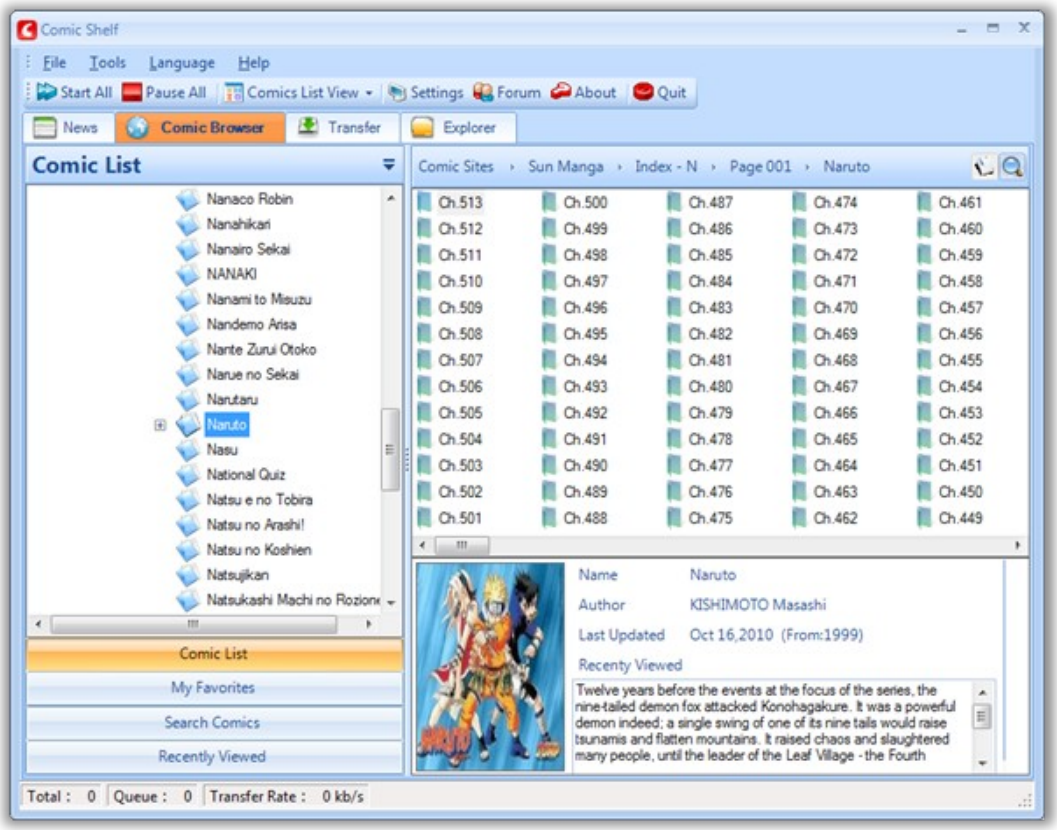

Ilustración 11: Comic Shelf 1 (Búsqueda de capítulos)

Escanea las webs de mangas creando una lista de mangas con sus capítulos presentes en cada web. Permite marcar como favoritos los mangas, pero no marca los capítulos que ya has leído. Permite abrir los capítulos directamente en un visor propio del programa o descargarlos, generando un archivo con todo el capítulo entero.

Ilustración 12: Comic Shelf 2 (Visor de capítulo)

La intefaz es un poco confusa y añade demasiadas opciones no relacionadas con la finalidad del programa en si. En cambio opciones importantes para la finalidad del programa, como el añadir un manga a favoritos, no son fáciles de encontrar.

2.4. Análisis de características

Aplicación	Búsqueda y descarga de capítulos	Gestión de favoritos	Gestión de capítulos leídos	Visor integrado	Sincronización con otros dispositivos
CDisplay	No	No	No	Sí	No
GonVisor	No	No	No	Sí	No
ComicRack	No	Sí	Sí	Sí	Sí
HakuNeko	Sí	No	No	No	No
Comic Shelf	Sí	Sí	No	Sí	No

Tabla 1: Análisis de características

2.5. Conclusiones

Tal y como se puede observar en el análisis de aplicaciones similares, no hay ninguna aplicación para escritorio que presente todas las características que se pretenden implementar en esta aplicación. La opción más completa probablemente sería combinar el uso de HakuNeko con ComiRack. Pero el tener que combinar dos aplicaciones dificulta el uso ya que no están integradas de ninguna forma. Por ejemplo HakuNeko no es capaz de decirte cuáles son los

capítulos que deberías descargar, ya que la información de capítulos de la librería y de capítulos leídos la tiene únicamente ComicRack. A pesar de esto también tienen otros problemas o carencias.

La mayoría de esta aplicaciones, presentan una interfaz confusa, con demasiadas opciones. Lo cual dificulta el uso de la aplicación. En la aplicación a desarrollar se implementará una interfaz simple, la cual en vez de ofrecer muchas opciones que confundan al usuario, establecerá por defecto las opciones que gusten a la mayoría y no desagraden a nadie. El facilitar el uso de la aplicación de esta manera, siendo fácil ver qué opciones te da la aplicación, y no ofreciendo opciones complejas, incrementa el uso. Como contra deben elegirse bien las opciones por defecto para que el usuario quede contento y no eche en falta el no poder establecer las opciones que quiera.

Por otra parte no hay ninguna que ofrezca una sincronización en la nube real. La única aplicación capaz de sincronizar sus datos entre distintos dispositivos es ComicRack, pero esta sólo es capaz de sincronizar la información desde el cliente de escritorio a los clientes móviles directamente. Si se desinstalara la aplicación de escritorio se perderían todos los datos. En la aplicación a desarrollar, en cambio, se pretende implementar una sincronización en la nube. Los usuarios se identifican por su correo y los datos de sus mangas favoritos y capítulos leídos se guardan en un servidor al momento y de forma transparente al usuario. Si tras esto el usuario abre la aplicación en otro dispositivo donde esté identificado con el mismo email, la aplicación automáticamente sincroniza toda su información de mangas favoritos y capítulos leídos con el servidor, mostrando al usuario lo que acaba de marcar en el otro dispositivo. Este método de sincronización en la nube también tiene la ventaja de que aunque desinstale completamente la aplicación de todos sus dispositivos, al estar sus datos guardados en la nube, al volver a instalarla y acceder con su email automáticamente volverá a tener disponibles todos sus datos. Por supuesto será opcional la sincronización en la nube, pudiendo usar la aplicación sin hacer ningún tipo de registro, pero permitiendo crear una cuenta fácilmente en cualquier momento y que todo lo que ha ido guardando antes de registrarse que se asocie a su cuenta automáticamente.

A parte, se implementará un visor a pantalla completa, simple pero con todas las opciones importantes. Permitirá descargas de páginas en paralelo, recargar páginas y varias opciones de visualización básicas como el zoom. Estas opciones serán accesibles mediante botones el la interfaz o mediante combinaciones de teclas intuitivas.

3. Especificación de requisitos

3.1. Introducción

En este apartado se describe la especificación de requisitos (ERS) de la aplicación de escritorio siguiendo la estructura definida y las prácticas recomendadas en el estándar IEEE 830-1998.

3.1.1. Propósito

Este apartado tiene como propósito definir los requisitos funcionales y no funcionales de la aplicación. Está dirigido al desarrollador de la aplicación, para que sirva de base en el desarrollo.

3.1.2. Ámbito del sistema

El objetivo será desarrollar el cliente de escritorio de la aplicación, la cuál recibirá el nombre de MangaYomu, así como el servidor que de soporte a las peticiones del cliente. El objetivo de esta aplicación será facilitar al usuario el acceso a los mangas, permitiéndoles buscarlos, leerlos, clasificarlos como favoritos y llevar la cuenta capítulos leídos en múltiples dispositivos.

3.1.3. Acrónimos

Nombre	Descripción
ERS	Especificación de Requisitos Software
RFXX	Se utilizará la siguiente numeración de los requisitos funcionales: R: Requisito F: Funczional XX: Número incremental
RNFXX	Se utilizará la siguiente numeración de los requisitos no funcionales: R: Requisito N: No F: Funcional XX: Número incremental

Tabla 2: Acrónimos

3.1.4. Definiciones

Nombre	Descripción
Manga	Cómic de origen japonés
REST	REpresentational State Transfer: Interfaz entre sistemas sobre HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos en cualquier formato sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes.
JSON	JavaScript Object Notation: Es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML, comúnmente usado para el intercambio de datos entre sistemas.
Cliente	Parte de la aplicación que se ejecuta en el ordenador del usuario.
Servidor	Parte de la aplicación que se ejecuta en la nube.

Tabla 3: Definiciones

3.1.5. Referencias

IEEE 830-1998: Recommended Practice for Software Requirements Specifications.

3.1.6. Visión general de la especificación de requisitos

El punto de la especificación de requisitos consta de tres secciones:

1. La primera sección (sección actual) realiza una breve introducción al apartado de la especificación de requisitos.
2. La segunda sección describe todos aquellos factores que afectan al producto y a sus requisitos. No se describen los requisitos, sino su contexto. Además proporciona una visión muy general del sistema.
3. Por último, la tercera sección contiene los requisitos a un nivel de detalle suficiente como para permitir a los diseñadores diseñar un sistema que satisfaga estos requisitos, y que permita al equipo de pruebas planificar y realizar las pruebas que demuestren si el sistema satisface, o no, los requisitos.

3.2. Descripción general

Antes de detallar los requisitos del sistema, es importante hacer una descripción del sistema de alto nivel y analizar aquellos factores que afectan al producto y a sus requisitos. Esta sección incluye: perspectiva del producto, funciones del producto, características de los usuarios, restricciones y suposiciones.

3.2.1. Perspectiva del producto

El producto final constará de tres partes:

1. Cliente móvil.
2. Cliente de escritorio
3. Servidor

El servidor actualizará su base de datos de mangas periódicamente a partir de las diversas fuentes programadas. También almacenará los favoritos y los capítulos leídos de los usuarios que se hayan identificado con su email.

Los clientes consultarán en el servidor la lista de mangas y sus capítulos, permitiendo a los usuarios leerlos. En caso de estar identificado, el usuario sincronizará los cambios con el servidor, de forma que este siempre tenga la misma información aunque cambie de dispositivo o de plataforma.

3.2.2. Funciones del producto

La aplicación permitirá al usuario leer y gestionar sus mangas, sincronizando los datos de favoritos y capítulos leídos en la nube en caso de estar identificado.

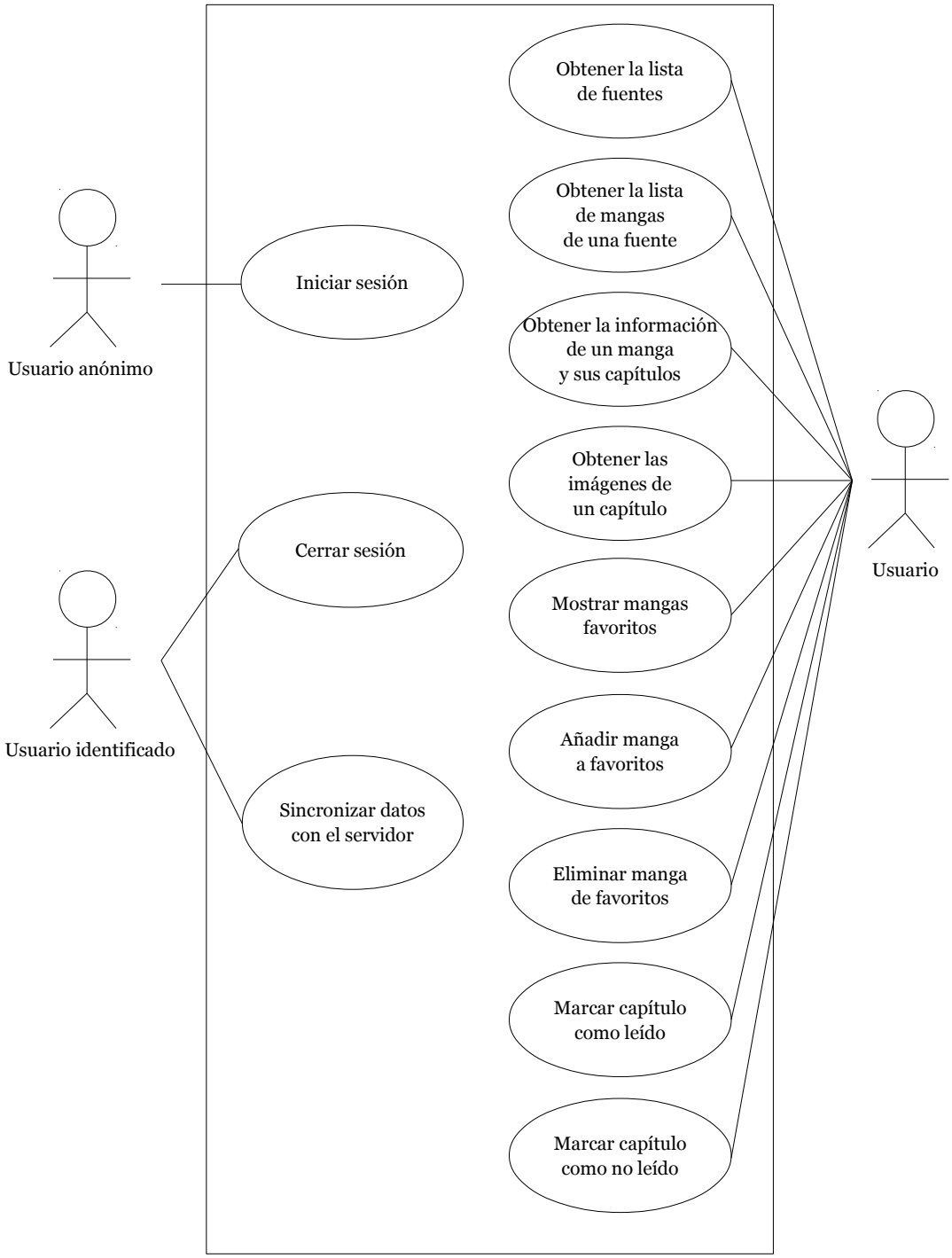

Ilustración 13: Diagrama de casos de uso

3.2.3. Características de los usuarios

La aplicación deberá poder ser usada por cualquier usuario estándar sin ningún tipo de conocimiento específico, por lo que deberá ser sencilla e intuitiva.

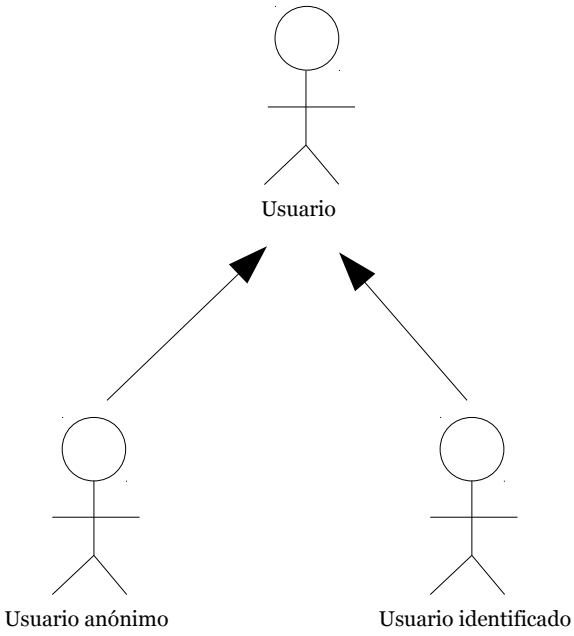

Ilustración 14: Diagrama de actores del sistema

Tipo de usuario	Usuario
Formación	Conocimientos básicos de utilización de programas informáticos
Actividades	Acceso a todas las funciones de lectura de mangas, gestión de favoritos y capítulos leídos.

Tabla 4: Características del actor Usuario

Tipo de usuario	Usuario anónimo
Formación	Conocimientos básicos de utilización de programas informáticos
Actividades	Iniciar sesión.

Tabla 5: Características del actor Usuario anónimo

Tipo de usuario	Usuario identificado
Formación	Conocimientos básicos de utilización de programas informáticos
Actividades	Sincronizar con la nube sus datos y cerrar sesión.

Tabla 6: Características del actor Usuario identificado

3.2.4. Restricciones

- La aplicación deberá poder ejecutarse sobre los principales sitemas operativos de escritorio (Windows, Linux, MacOS).
- El cliente se escribirá mediante el framework Qt sobre C++.
- El servidor utilizará el frameword Sinatra sobre Ruby.
- La base de datos del servidor será una instancia de MongoDB.
- Se utilzará REST para la comunicación entre el cliente y el servidor.

3.2.5. Suposiciones y dependencias

- El dispositivo donde se ejecute la aplicación tendrá conexión a internet.
- Se utilizará un dominio fijo para identificar al servidor, el cual estará disponible públicamente.
- El servidor se ejecutará sobre un puerto fijo el cual podrá ser accedido desde una red externa.

3.3. Requisitos específicos

Esta sección detalla los requisitos funcionales y no funcionales con suficiente nivel de detalle para que el equipo de desarrollo pueda cubrir y comprobar todas las necesidades expuestas por el cliente. Todos los requisitos tendrán que ser cubiertos por el sistema.

3.3.1. Interfaces externas

- Intefaz de usuario: Los usuario podrán manejar la aplicación a través de la interfaz de usuario.
- Interfaz REST: La aplicación podrá comunicarse con el servidor mediante el envío de mensajes a través de la red.
- Interfaz HTTP: El servidor se comunicará mediante peticions HTTP con webs externas para obtener periódicamente la lista de mangas y capítulos de cada web.

3.3.2. Requisitos funcionales

Identificador	RF01
Nombre	Iniciar sesión.
Descripción	Al usuario se le pedirá su email. Si no existe ninguna cuenta con ese email se le pedirá además un nombre de usuario. Tras esto se le enviará un email con un código que debe introducir.
Entradas	Email y código.
Salidas	Si introduce un email con formato incorrecto se le informará de ello. Si el email introducido no está usado se le informará de que adicionalmente debe proporcionar un nombre de usuario. Si el nombre de usuario ya está usado se le informará de esta situación. Si el email y código son correctos se producirá el inicio de sesión.

Tabla 7: Requisito funcional 1 (Iniciar sesión)

Identificador	RF02
Nombre	Cerrar sesión.
Descripción	Al pulsar el botón de cerrar sesión el cliente esperará a que se terminen de ejecutar las peticiones pendientes al servidor y tras esto enviará una petición al servidor para que este borre los datos de la sesión y borrará los datos de la sesión en el cliente.
Entradas	Pulsación en el botón de cerrar sesión.
Salidas	En caso de error se mostrará el error.

Tabla 8: Requisito funcional 2 (Cerrar sesión)

Identificador	RF03
Nombre	Sincronizar datos con el servidor.
Descripción	Tras iniciar sesión y siempre que se inicie la aplicación teniendo una sesión activa, se llevará a cabo este proceso. Cuando esto ocurra se descargará del servidor toda la información de favoritos y capítulos leídos del usuario y se actualizará la información del usuario con estos datos. En el caso de ser un usuario nuevo también subirá al servidor los datos de favoritos y capítulos leídos que estén en el cliente.
Entradas	Ninguna.
Salidas	Se mostrará un icono en la aplicación que indique que hay una sincronización activa.

Tabla 9: Requisito funcional 3 (Sincronizar datos con el servidor)

Identificador	RF04
Nombre	Obtener la lista de fuentes.
Descripción	Al iniciar la aplicación, conectará con el servidor para actualizar la lista de fuentes de mangas. Esta lista deberá quedar guardada en el cliente para que la siguiente vez que se inicie pueda mostrar una lista desde el principio y permitiendo utilizar la aplicación. Una vez se descargue la la información de las nuevas fuentes desde el servidor esta lista se actualizará con los nuevos datos.
Entradas	Ninguna.
Salidas	Se muestra la lista de fuentes disponibles.

Tabla 10: Requisito funcional 4 (Obtener la lista de fuentes)

Identificador	RF05
Nombre	Obtener la lista de mangas de una fuente.
Descripción	Al pulsar el una fuente en la lista de fuentes, descargará del servidor la lista de mangas de esa fuente. El cliente deberá guardar las listas de mangas que ya haya descargado hasta que se cierre el programa para evitar descargar varias veces la misma lista cuando el usuario cambia de una fuente a otra repetidamente.
Entradas	Ninguna.
Salidas	Se muestra la lista de mangas disponibles en esa fuente.

Tabla 11: Requisito funcional 5 (Obtener la lista de mangas de una fuente)

Identificador	RF06
Nombre	Obtener la información de un manga y sus capítulos.
Descripción	Al pulsar sobre un manga se descargará del servidor la información del manga y su lista de capítulos.
Entradas	Ninguna.
Salidas	Se mostrará en pantalla el nombre del manga, la fuente de la que se ha obtenido la información del manga, la imagen si tuviera una, la descripción y la lista de capítulos.

Tabla 12: Requisito funcional 6 (Obtener la información de un manga y sus capítulos)

Identificador	RF07
Nombre	Obtener las imágenes de un capítulo.
Descripción	Al pulsar sobre un capítulo descargará las imágenes del capítulo y las mostrará en un visor a pantalla completa que permitirá pasar de imagen hacia delante o hacia atrás.
Entradas	Ninguna.
Salidas	Visor de imágenes a pantalla completa con las imágenes del capítulo.

Tabla 13: Requisito funcional 7 (Obtener las imágenes de un capítulo)

Identificador	RF08
Nombre	Mostrar mangas favoritos.
Descripción	Al pulsar sobre la opción de favoritos en la vista principal mostrará una lista de todos los mangas favoritos.
Entradas	Ninguna.
Salidas	La lista de mangas favoritos, en la que para cada elemento de la lista se mostrará el nombre del manga, la cantidad de capítulos leídos, la cantidad total de capítulos y la imagen del manga si tuviera una.

Tabla 14: Requisito funcional 8 (Mostar mangas favoritos)

Identificador	RF09
Nombre	Añadir manga a favoritos.
Descripción	Al pulsar el botón “Añadir a favoritos” en la vista de la información de un manga que no es favorito, este se añadirá a la lista de favoritos. Si el usuario estuviera identificado, además, se enviará una petición al servidor para añadir el favorito y su lista de capítulos leídos a la información del usuario
Entradas	Ninguna.
Salidas	El botón “Añadir a favoritos” cambiará por el botón “Eliminar de favoritos”.

Tabla 15: Requisito funcional 9 (Añadir manga a favoritos)

Identificador	RF10
Nombre	Eliminar manga de favoritos.
Descripción	Al pulsar sobre el botón de “Eliminar de favoritos” en la vista de la información de un manga que es favorito, este se eliminará de la lista de favoritos. Si el usuario estuviera identificado, además, se enviará una petición al servidor para eliminar el manga de favoritos, eliminando con ello del servidor la lista de capítulos leídos también.
Entradas	Ninguna.
Salidas	El botón “Eliminar de favoritos” cambiará por el botón “Añadir a favoritos”.

Tabla 16: Requisito funcional 10 (Eliminar manga de favoritos)

Identificador	RF11
Nombre	Marcar capítulo como leído.
Descripción	Al pulsar sobre el botón “Marcar capítulo como leído” en la lista de capítulos de un manga junto a un capítulo que no estuviera leído, el capítulo se marcará como leído. Si el usuario estuviera identificado y el capítulo perteneciera a un manga marcado como favorito, además, se hará una petición al servidor para que marque el capítulo como leído. También se lanzará esta acción al llegar a la última página en la vista de lectura de un capítulo.
Entradas	Ninguna.
Salidas	El botón de “Marcar capítulo como leído” cambiará por el de “Marcar capítulo como no leído” y el nombre del capítulo aparecerá como deshabilitado en la lista de capítulos.

Tabla 17: Requisito funcional 11 (Marcar capítulo como leído)

Identificador	RF12
Nombre	Marcar capítulo como no leído.
Descripción	Al pulsar sobre el botón “Marcar capítulo como no leído” en la lista de capítulos de un manga junto a un capítulo que estuviera leído, el capítulo se marcará como no leído. Si el usuario estuviera identificado y el capítulo perteneciera a un manga marcado como favorito, además, se hará una petición al servidor para que marque el capítulo como no leído.
Entradas	Ninguna.
Salidas	El botón de “Marcar capítulo como no leído” cambiará por el de “Marcar capítulo como leído” y el nombre del capítulo aparecerá como habilitado en la lista de capítulos.

Tabla 18: Requisito funcional 12 (Marcar capítulo como no leído)

3.3.3. Requisitos no funcionales

Identificador	RNF01
Tipo	Seguridad.
Nombre	Control de acceso.
Descripción	Para sincronizar los datos de usuario en el servidor el usuario deberá haberse identificado correctamente como propietario del email introduciendo un código que se le enviará a la dirección que ha indicado. Una vez identificado sólo podrá modificar los datos del servidor pertenecientes a su propio usuario.

Tabla 19: Requisito no funcional 1 (Control de acceso)

Identificador	RNF02
Tipo	Seguridad.
Nombre	Identificación segura de la sesión.
Descripción	Para las peticiones de sincronización de datos del usuario se requerirá tener una sesión iniciada. La identificación de esta sesión se realizará mediante un token de sesión y un código de identificación que se obtendrá de aplicar el algoritmo MD5 sobre la concatenación del código PIN enviado al email del usuario al iniciar sesión y un número aleatorio controlado por el servidor. El código PIN en ningún momento deberá transferirse de ninguna forma que no sea codificado de esta forma entre el servidor y el cliente o viceversa. El servidor decidirá el numero aleatorio a concatenar para generar el código de identificación la siguiente petición y se lo comunicará al cliente al responder a la petición anterior. Si el cliente envía un código de identificación incorrecto el servidor le responderá con un error volviéndole a enviar el mismo número aleatorio hasta que acierte el número. El servidor deberá evitar generar dos veces el mismo número aleatorio para evitar que se pueda reutilizar el mismo código de identificación dos veces.

Tabla 20: Requisito no funcional 2 (Identificación segura de la sesión)

Identificador	RNF03
Tipo	Fiabilidad.
Nombre	Control de errores.
Descripción	Tanto el servidor como el cliente deberán manejar los errores para evitar comportamientos inesperados en la aplicación. El servidor deberá transformar cualquier posible error conocido a una respuesta con un código que la aplicación sea capaz de entender y manejar. La aplicación deberá estar preparada para ante cualquier error inesperado como problemas con la red u otra cosa sea capaz de continuar su ejecución lanzando reintentos de sincronización con el servidor si hiciera falta para no perder el estado, o mostrando un mensaje de error afectara a la experiencia de usuario y no fuera recuperable.

Tabla 21: Requisito no funcional 3 (Control de errores)

Identificador	RNF04
Tipo	Operatividad.
Nombre	Disponibilidad del servidor.
Descripción	El servidor debe estar siempre accesible 24 hora los 365 días del año.

Tabla 22: Requisito no funcional 4 (Disponibilidad del servidor)

4. Diseño del sistema

4.1. Introducción

En este apartado se especifica el diseño tanto formal como conceptual del sistema. En el diseño conceptual se describe la arquitectura del sistema. En el diseño formal, en cambio, se incluyen los prototipos de la aplicación, diagramas de flujo, modelos de la base de datos y la API de comunicación con el servidor.

4.2. Diseño conceptual del sistema

4.2.1. Arquitectura

Para este proyecto se ha optado por una arquitectura cliente-servidor.

El servidor se encargará de obtener periódicamente datos de las distintas fuentes de mangas y ofrecerá estos datos al cliente. También guardará la lista de mangas favoritos y sus capítulos leídos para usuarios identificados.

El cliente obtendrá toda la información necesaria sobre mangas y capítulos del servidor. Lo único que buscará en fuentes externas serán las imágenes de las páginas de los capítulos, cuyas urls le indicará el servidor. Independientemente de si el usuario está identificado o no, el cliente guardará localmente la lista de favoritos y el estado de lectura de todos los capítulos pertenezcan a mangas favoritos o no.

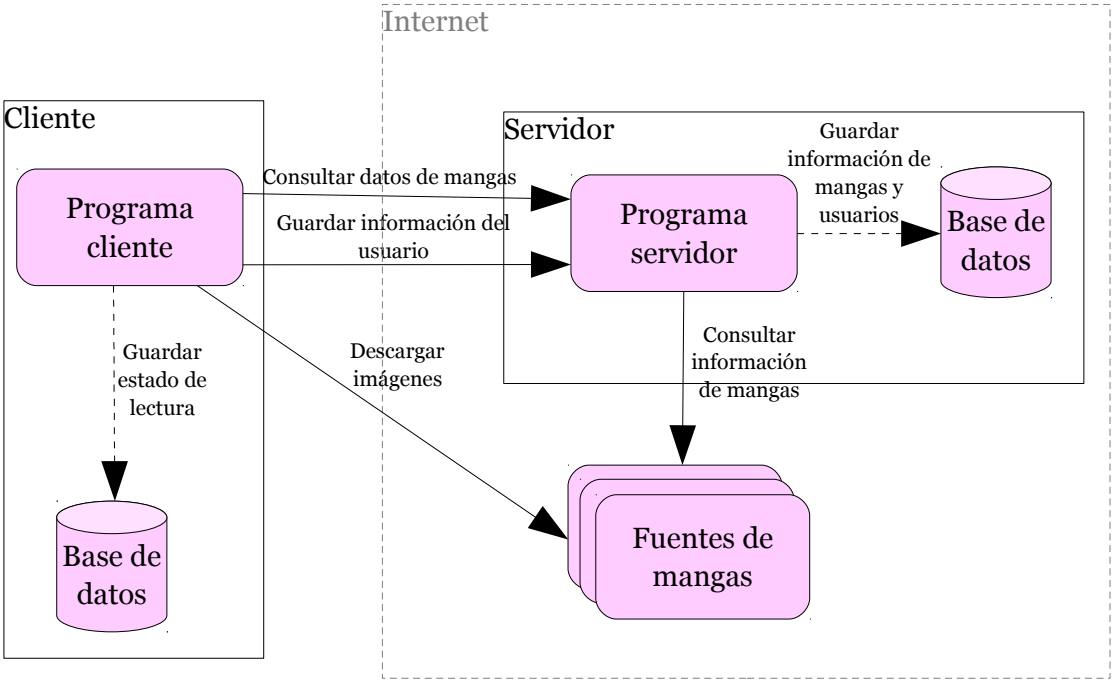

Ilustración 15: Diseño de la arquitectura del sistema

4.3. Diseño formal del sistema

4.3.1. Prototipos

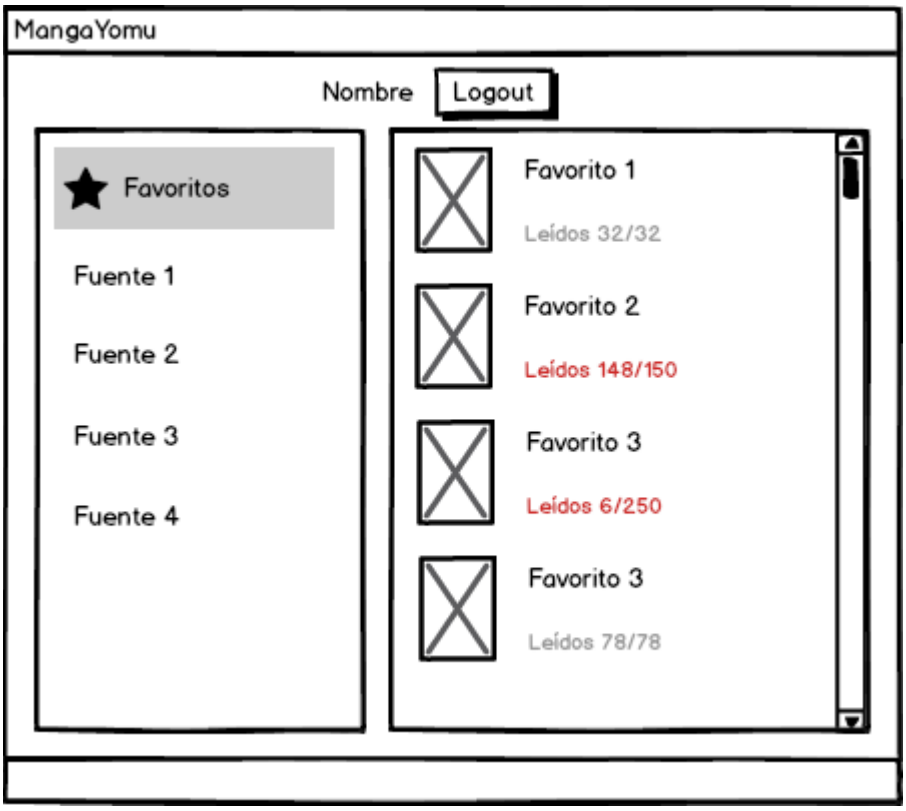

Ilustración 16: Mockup (Favoritos)

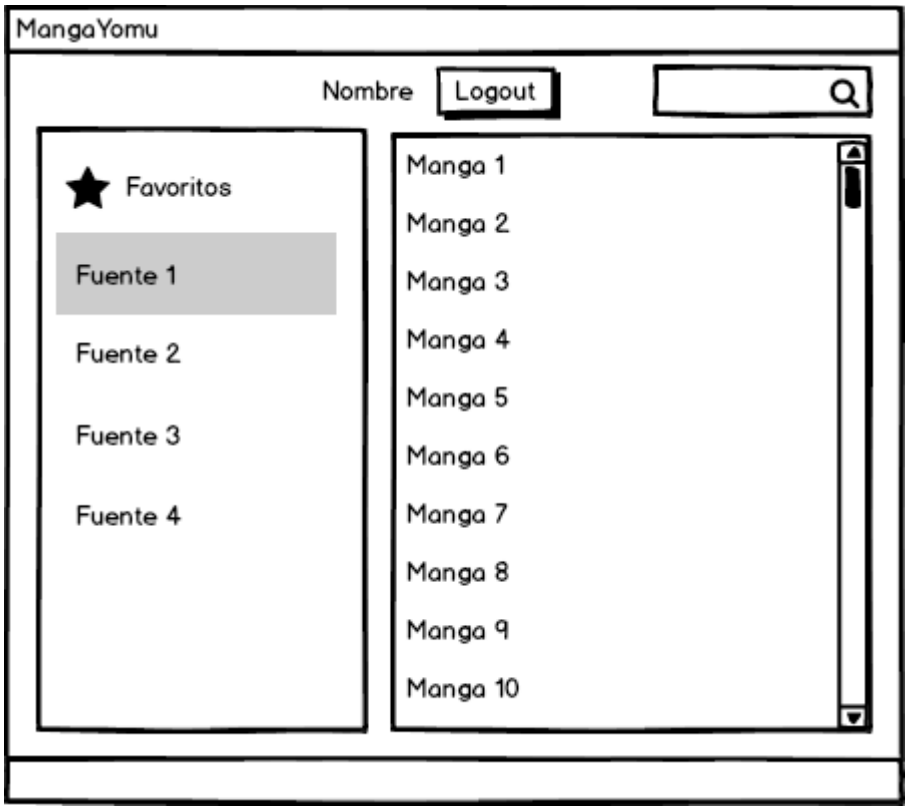

Ilustración 17: Mockup (Mangas de una fuente)

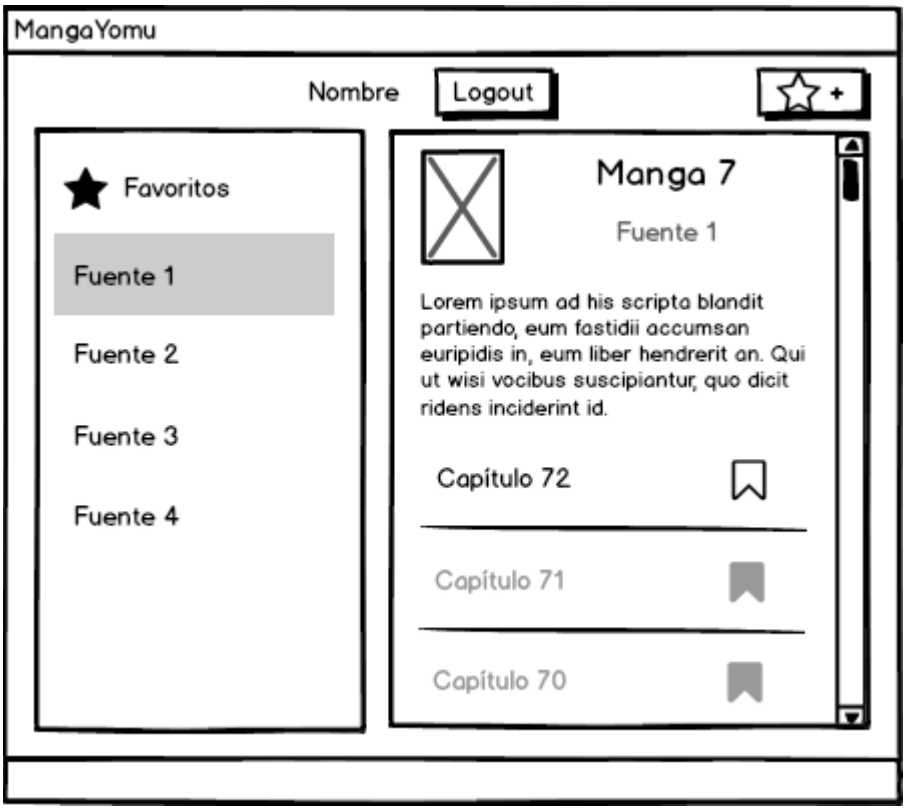

Ilustración 18: Mockup (Información y capítulos de un manga)

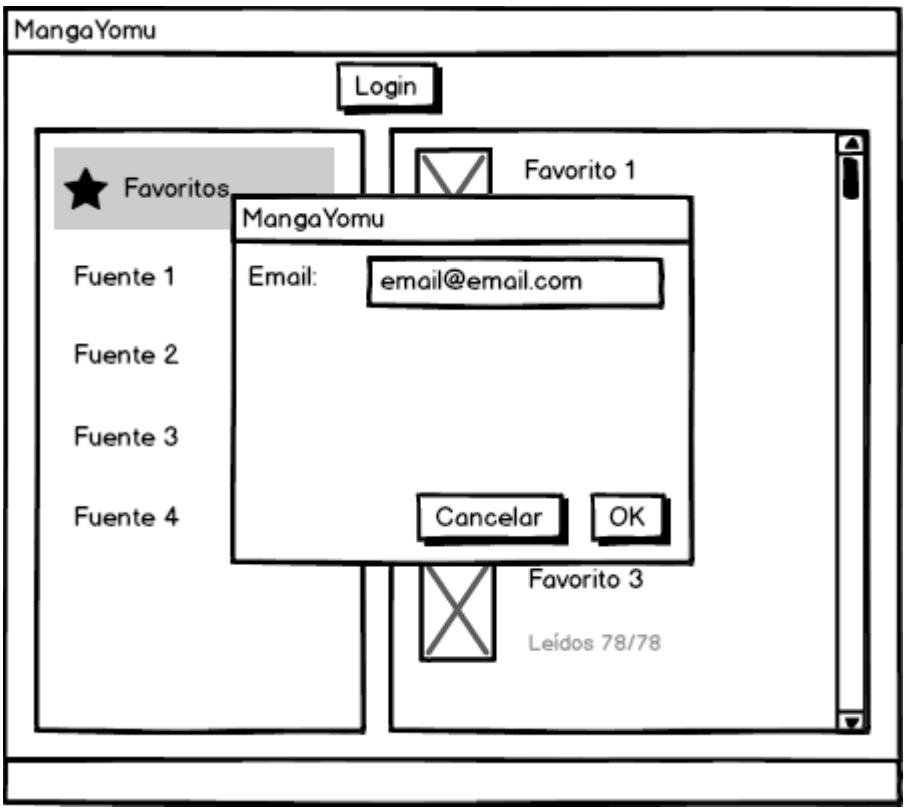

Ilustración 19: Mockup (Iniciar sesión: Introducir email)

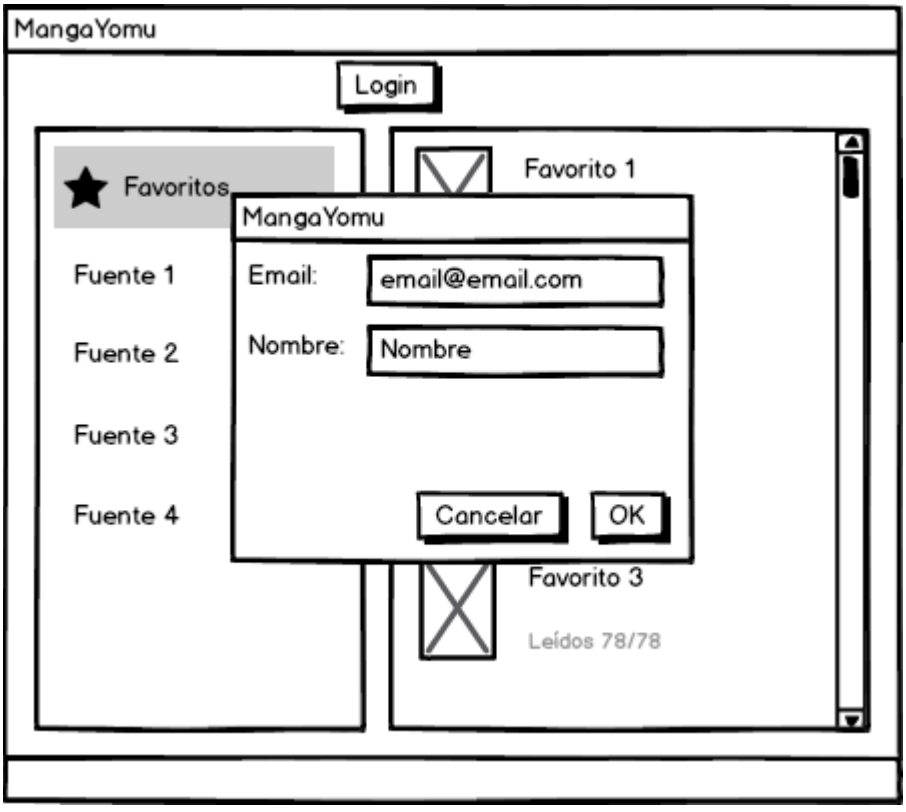

Ilustración 20: Mockup (Iniciar sesión: Introducir nombre)

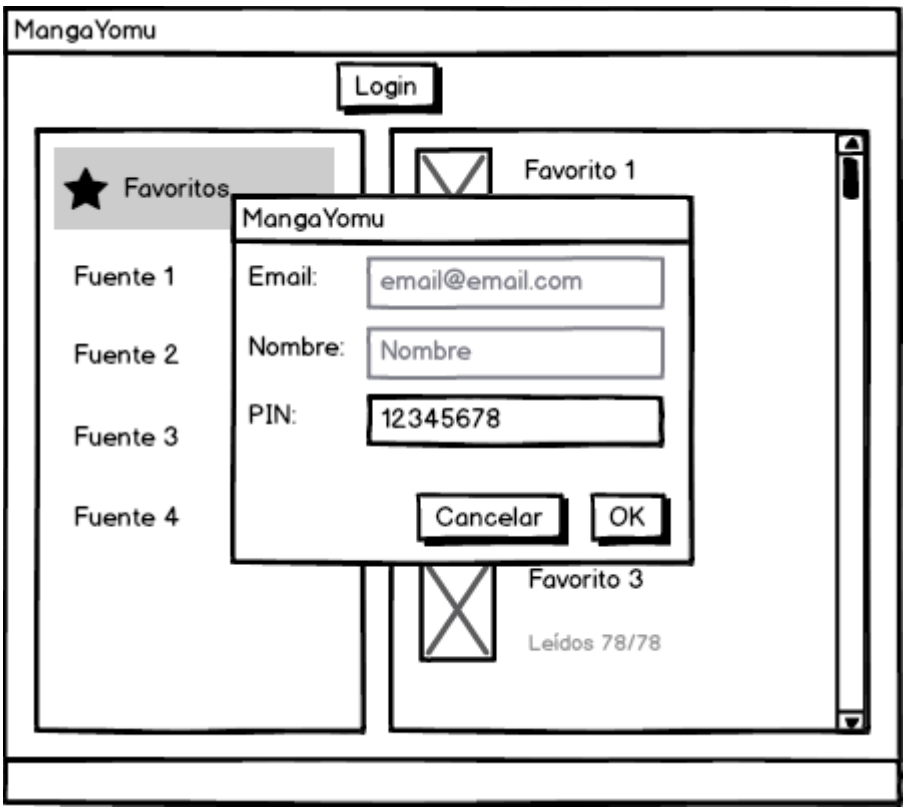

Ilustración 21: Mockup (Iniciar sesión: Introducir PIN)

Ilustración 22: Mockup (Visor de capítulo: Cargando página)

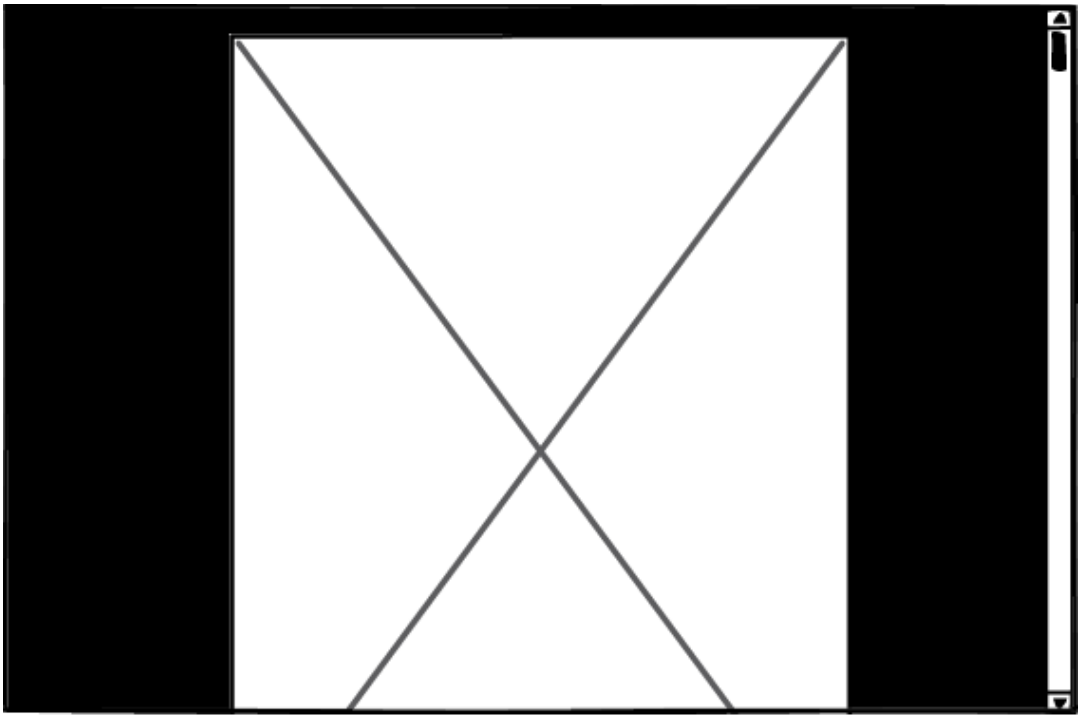

Ilustración 23: Mockup (Visor de capítulo: Inicio de la página)

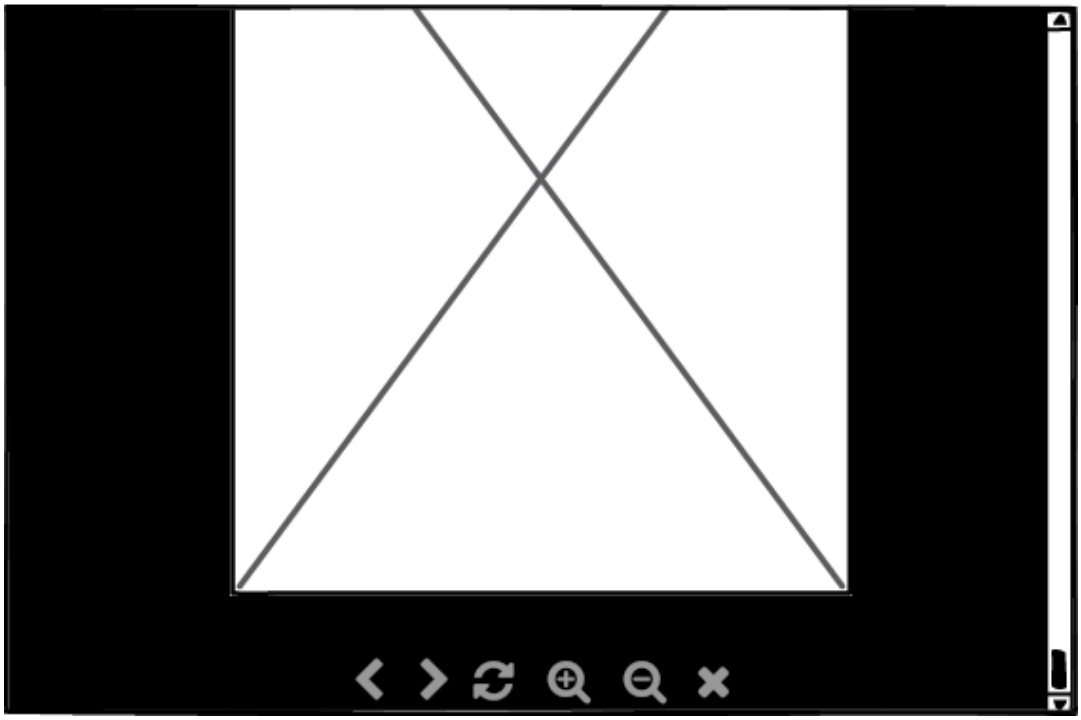

Ilustración 24: Mockup (Visor de capítulo: Final de la página)

Ilustración 25: Mockup (Visor de capítulo: Final del capítulo)

4.3.2. Diagramas de flujo

4.3.2.1. Iniciar sesión

Al iniciar sesión primero debe pedir el email del usuario. Si nunca ha sido usado se creará un nuevo usuario con ese email y para ello se pedirá un nombre para el nuevo usuario el cual no debe haber sido usado por otro usuario.

Tras esto se enviará un correo a la dirección de email introducida con un PIN y se le responderá al cliente con la clave de sesión y el número aleatorio para cifrar la próxima petición. El cliente pedirá al usuario que introduzca el PIN recibido y enviará una petición al servidor con la clave generada a partir del PIN para comprobar que es correcto. No le dejará continuar hasta que introduzca el PIN correcto.

Si en cualquier petición al servidor, este responde con un mensaje desconocido u ocurre cualquier error de red que impida la comunicación, se mostrará un mensaje de error. Además se permitirá cancelar el inicio cerrando la ventana de inicio de sesión en cualquier punto del proceso.

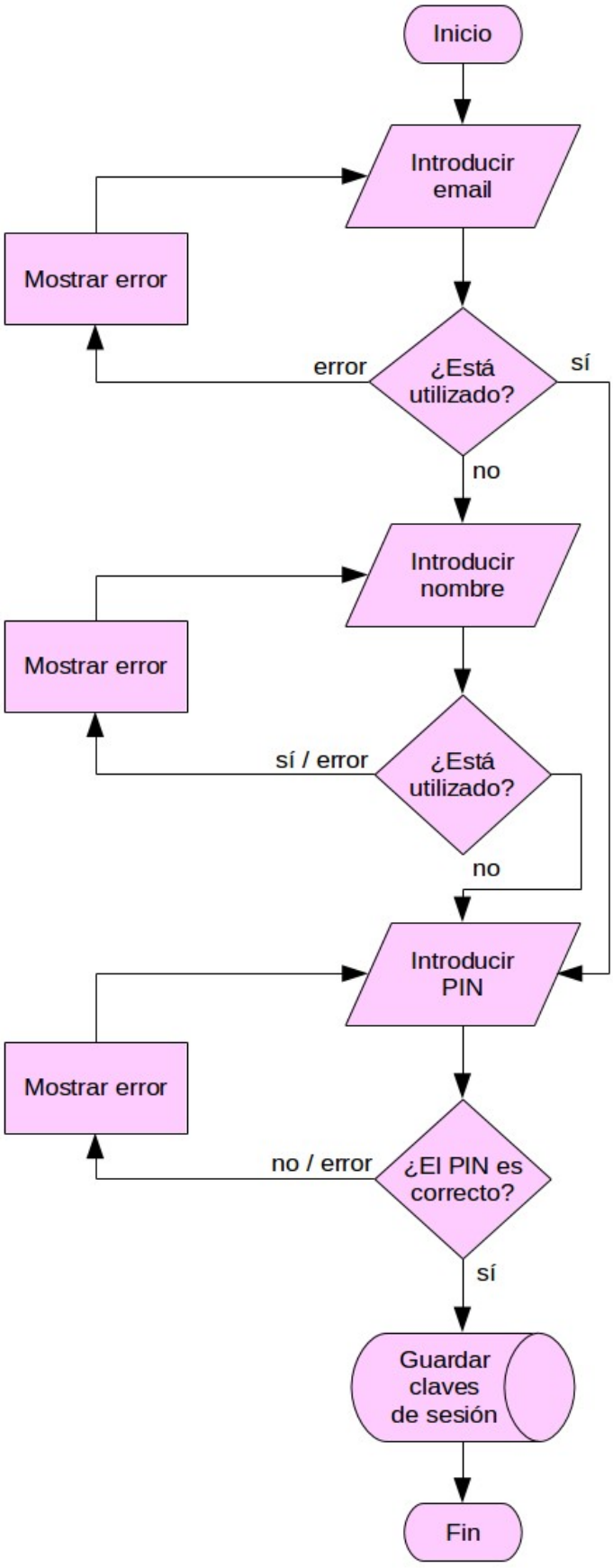

Ilustración 26: Diagrama de flujo (Iniciar sesión)

4.3.2.2. Cerrar sesión

Primero envía una petición al servidor para que borre los datos de la sesión, y si el resultado es satisfactorio borra los datos de la sesión en el cliente.

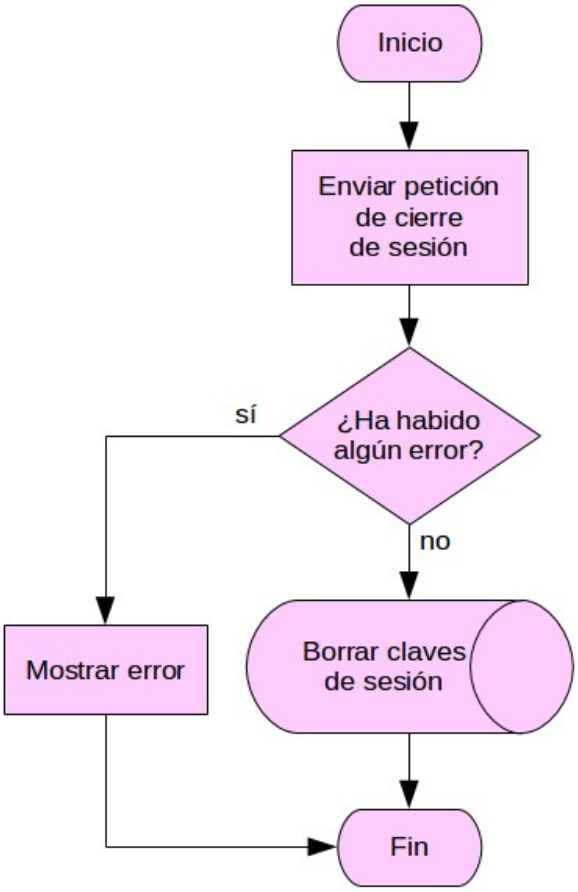

Ilustración 27: Diagrama de flujo (Cerrar sesión)

4.3.2.3. Sincronizar datos de usuario

Esta acción se lanza tras iniciar sesión o al iniciar el programa teniendo una sesión iniciada.

Para llevar esto a cabo, primero se descargan los datos de favoritos y capítulo leídos del servidor, luego se mezclan los datos con los datos locales, dando prioridad a los datos del servidor. Cuando se ejecuta tras iniciar sesión además puede guardar datos locales en el servidor dependiendo de ciertas condiciones.

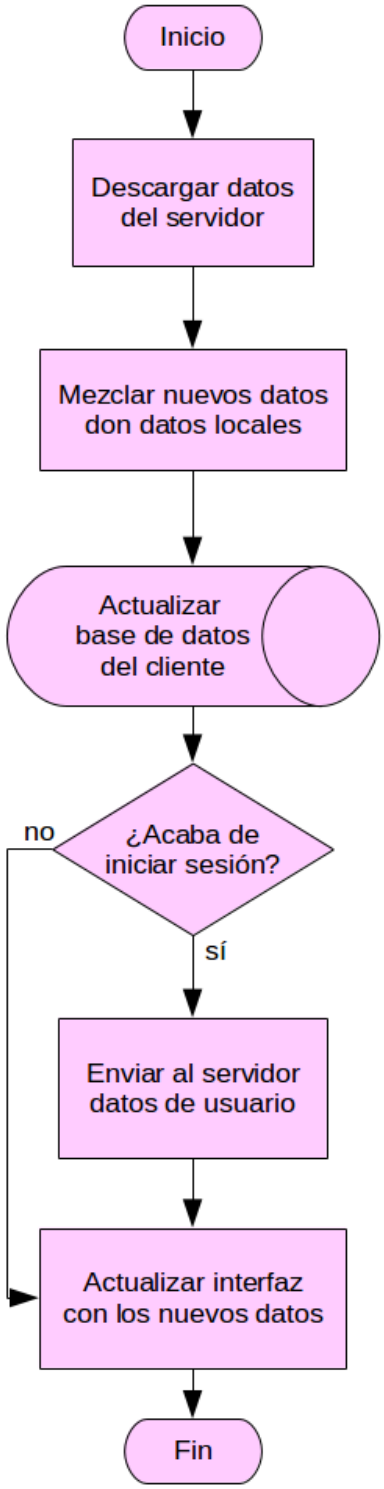

Ilustración 28: Diagrama de flujo (Sincronizar datos de usuario)

4.3.2.4. Añadir manga a favoritos

Al añadir un favorito, además de marcarlo en el propio cliente, si el usuario tiene una sesión iniciada, se deben enviar los datos al servidor para que se quede marcado también en la nube. Puesto que en el servidor sólo se guardan los

capítulos leídos de los mangas que son favoritos, mientras que en el cliente se guardan los capítulos leídos de cualquier manga, al enviar un favorito al servidor deben enviarse junto a este todos los capítulos leídos de este manga. De esta forma quedan todos los datos de ese manga registrados a partir de ese momento.

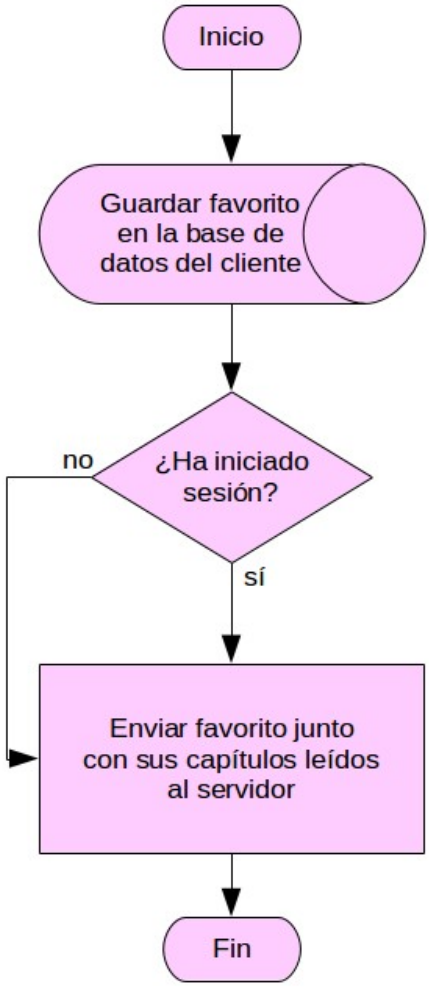

Ilustración 29: Diagrama de flujo (Añadir manga a favoritos)

4.3.2.5. Quitar manga de favoritos

La acción es equivalente a la de marcar como favoritos, pero no se envía ningún dato de los capítulos leídos, los cuales serán borrados del servidor. Pero, en cambio, el cliente sí guardará el estado de lectura de los capítulos, volviendo a enviarlos al servidor si el manga volviera a ser marcado como leído habiendo una sesión activa.

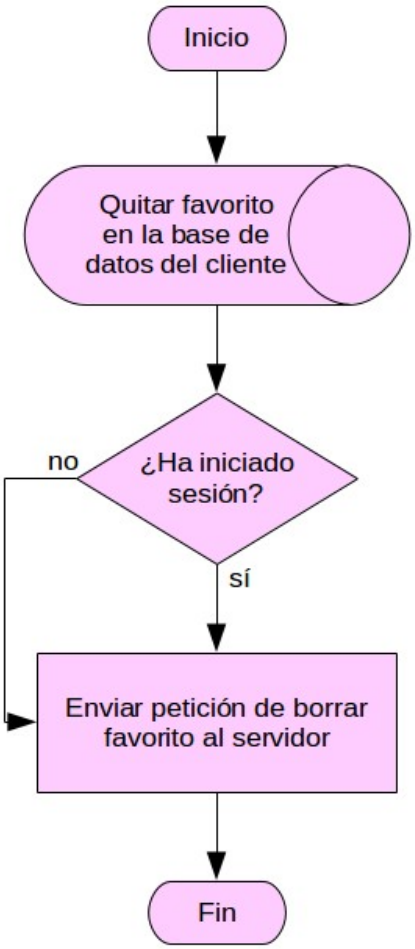

Ilustración 30: Diagrama de flujo (Quitar manga de favoritos)

4.3.2.6. Marcar o desmarcar capítulo como leído

Tras marcar/desmarcar el capítulo en local, en caso de ser un favorito y el usuario tener una sesión iniciada, se debe enviar esta acción al servidor para que se realice la misma acción y así tener sincronizados los datos del cliente y la nube.

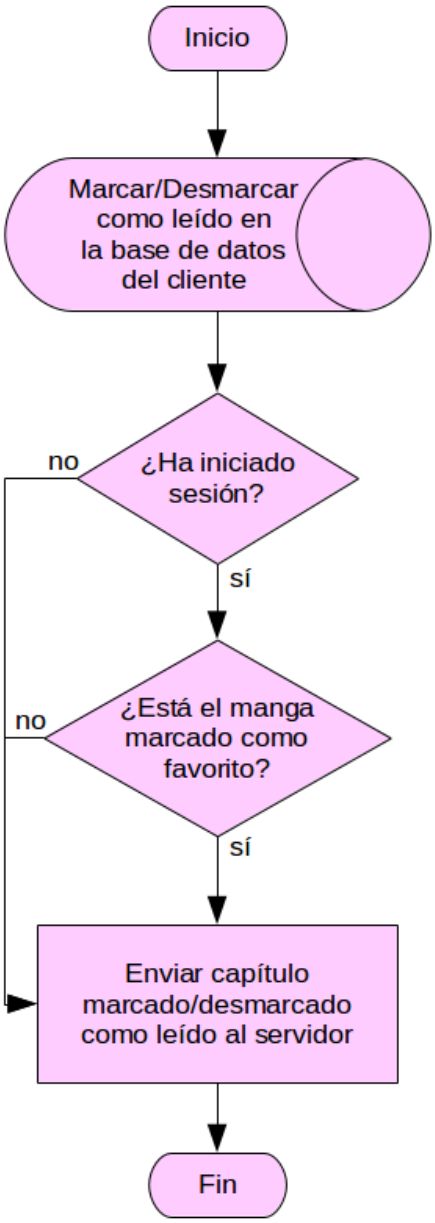

Ilustración 31: Diagrama de flujo (Marcar o desmarcar capítulo como leído)

4.3.2.7. Descargar página

Al abrir un capítulo o al darle a recargar en la visualización de una página se lanzará un proceso de descargar página. Este proceso pondrá a descargar en segundo plano la imagen y mientras irá mostrando en la pantalla el proceso de descarga en el lugar donde debería mostrarse la página.

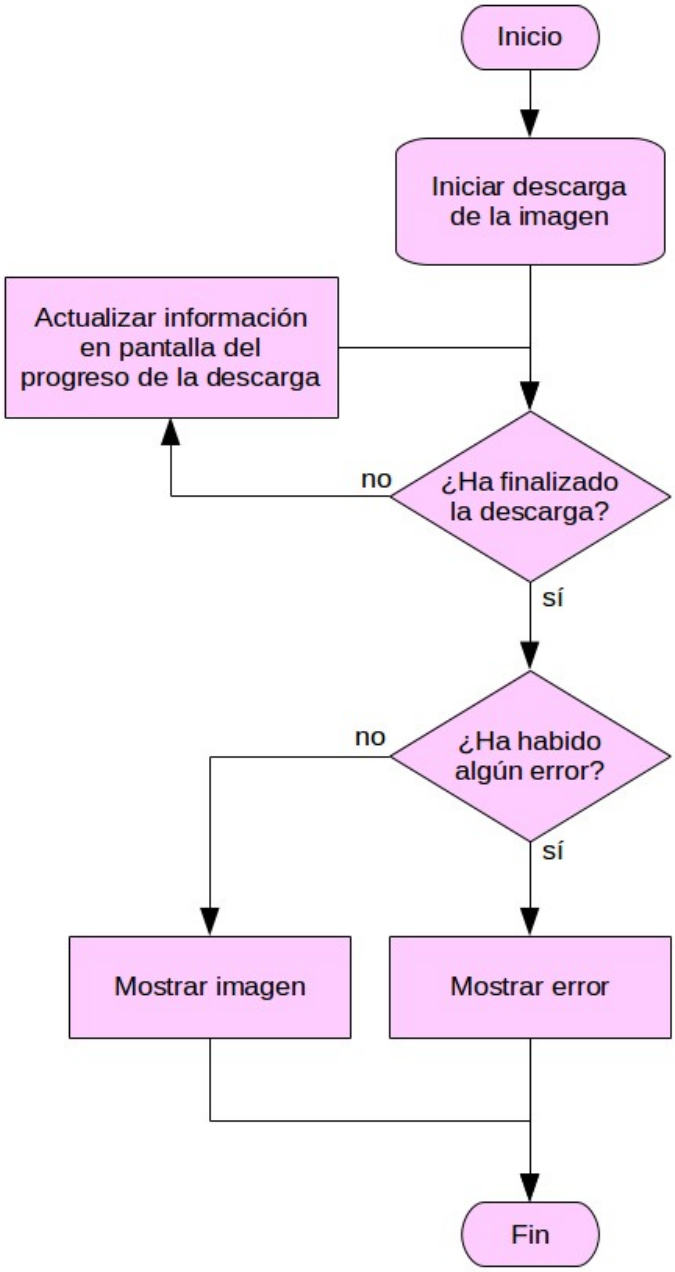

Ilustración 32: Diagrama de flujo (Descargar página)

4.3.3. Base de datos

4.3.3.1. Base de datos del cliente de escritorio

Para guardar el estado de la aplicación, como los capítulos leídos y otros datos, el cliente de escritorio utilizará una base de datos. En este caso se utilizará SQLite versión 3 o superior, proporcionada por el driver "QSQLITE" de Qt.

Esta es una base de datos relacional que no requiere la ejecución de un servidor, ejecutándose como una librería más del cliente y guardando los datos en un archivo en la carpeta del sistema que se indique.

Se ha intentado mantener la base de datos lo más sencilla posible para permitir inserciones y borrados rápidos. Además sólo guarda justo lo necesario para soportar las características del cliente.

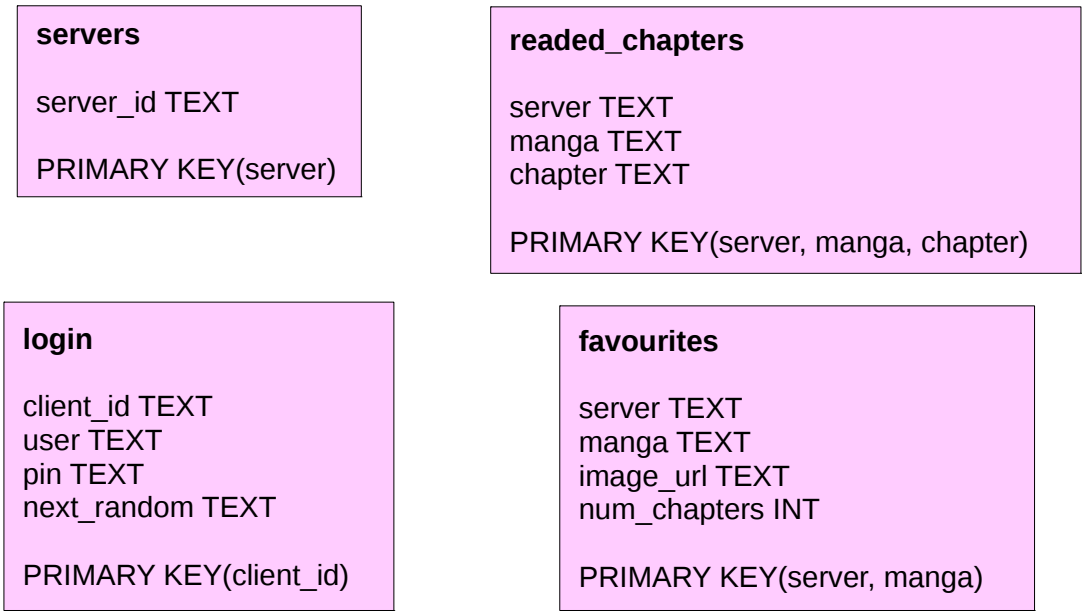

Ilustración 33: Modelo de base de datos (Cliente de escritorio)

Las tablas son independientes, no se utilizan claves externas, ya que intentar relacionarlas requeriría añadir tablas extras con información redundante. Además, facilita evitar posibles errores al intentar mantener la información sincronizada con el servidor, ya que el servidor puede mandar datos de mangas desconocidos para el cliente o borrar mangas sin avisar al cliente. De esta forma la base de datos queda más sencilla y robusta. De esta forma, un fallo temporal en el servidor que haga desaparecer un manga, no provocará un borrado en cascada en todos los clientes que hará desaparecer la información de los capítulos leídos de este manga.

4.3.3.2. Base de datos del servidor

La base de datos del servidor debe guardar la información de todos los mangas de cada fuente y también los favoritos y capítulos leídos de los usuarios que hayan iniciado sesión. Esta vez se ha utilizado la base de datos no relacional MongoDB para soportar los datos. Esta base de datos nos proporciona flexibilidad, algo muy necesario al obtener información de fuentes muy cambiantes, rapidez y una alta escalabilidad.

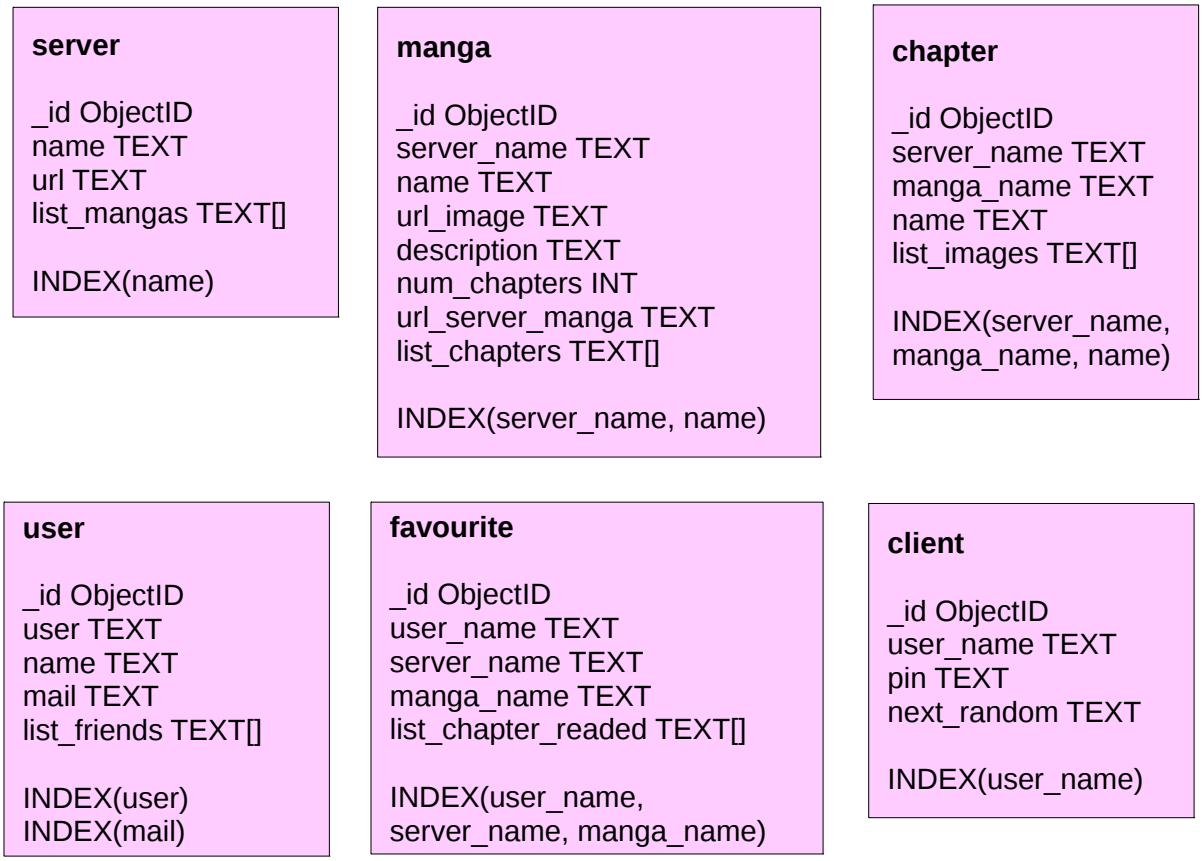

Ilustración 34: Modelo de base de datos (Servidor)

Nos aprovechamos de la flexibilidad de MongoDB para generar unas tablas, todas ellas independientes entre si, que permitan guardar y acceder fácilmente y de forma eficiente la información. De esta forma para obtener cualquier información, sólo se necesita acceder a un documento y devolver el contenido.

Por ejemplo, si se quiere la lista de mangas de capítulos de un manga, sólo hay que buscar en la tabla de mangas el manga que se quiere utilizar, y la propia entrada contiene la lista de capítulos de ese manga. Además todas las tablas tienen índices en las búsquedas más comunes los cuales permiten encontrar el documento mucho más rápido.

El guardar de esta forma los datos, sin relacionar las tablas, nos permite también excluir fácilmente, por ejemplo, mangas de la lista de un servidor sin borrar las entradas con la información de los mangas, de forma que si volvieran a estar disponibles simplemente se volvería añadir su nombre a la lista de mangas en la entrada del servidor y volvería a poderse acceder a toda su información. Esto también nos evita perder información y tener que volver a procesar un manga entero si por un fallo en la fuente momentáneamente desaparecieran mangas o capítulos de la fuente.

4.3.4. API de comunicación

Aquí se describen las diferentes llamadas que puede realizar el cliente al servidor para obtener datos o sincronizar información. Esta API se implementará utilizando REST, por lo que todo serán peticiones HTTP que incluirán los parámetros codificados como url en el path o en en las llamadas que lo permitan el cuerpo de la petición. El servidor devolverá una respuesta con el código de error adecuado si algo fallara, o en caso de ir todo bien el código de error 200 y los datos en el cuerpo de la respuesta codificados en JSON.

Nombre	Iniciar sesión
Descripción	Crea una sesión para el usuario, creando el usuario también si este no existiera.
Tipo	POST
Ruta	/user
Parámetros	user_name: string mail: string
Respuesta	user_name: string client_id: string next_random: string
Códigos de error	400: El usuario no existe, debe especificarse también el nombre de usuario. 403: Ya existe otro usuario con ese nombre, debe elegirse otro. 500: El email tiene un formato inválido.

Tabla 23: API REST (Iniciar sesión)

Nombre	Comprobar sesión
Descripción	Sirve para comprobar si la sesión se ha iniciado correctamente.
Tipo	GET
Ruta	/test_login
Parámetros	client_id: string key: string
Respuesta	user_name: string next_random: string
Códigos de error	400: Los datos de sesión son incorrectos y el usuario debe hacer login de nuevo. 401: La clave generada a partir del PIN y el número aleatorio no coincide.

Tabla 24: API REST (Comprobar sesión)

Nombre	Cerrar sesión
Descripción	Borra los datos de la sesión.
Tipo	GET
Ruta	/logout
Parámetros	client_id: string key: string
Respuesta	user_name: string
Códigos de error	400: Los datos de sesión son incorrectos y el usuario debe hacer login de nuevo. 401: La clave generada a partir del PIN y el número aleatorio no coincide.

Tabla 25: API REST (Cerrar sesión)

Nombre	Añadir manga a favoritos
Descripción	Añade un manga junto con la lista de capítulos ya leídos a la lista de favoritos del usuario indicado.
Tipo	POST
Ruta	/favourite/{user}
Parámetros	client_id: string key: string user: string (parámetro enviado mediante la ruta) server: string manga: string chapter: string | string[ ] (parámetro opcional)
Respuesta	next_random: string
Códigos de error	400: Los datos de sesión son incorrectos y el usuario debe hacer login de nuevo. 401: La clave generada a partir del PIN y el número aleatorio no coincide. 403: El usuario que ha iniciado la sesión no coincide con el usuario al que se quiere acceder.

Tabla 26: API REST (Añadir manga a favoritos)

Nombre	Obtener lista de favoritos
Descripción	Devuelve la lista de favoritos de un usuario. Opcionalmente puede devolver también la lista de capítulos leídos de cada favorito, si así se indica mediante el parámetro “addreaded”. También puede añadir el número de capítulos y la url de la imagen a la información de cada favorito si así se indica mediante el parámetro “addextrainfo”.
Tipo	GET
Ruta	/favourite/{user}
Parámetros	client_id: string key: string user: string (parámetro enviado mediante la ruta) addreaded: bool (parámetro opcional) addextrainfo: bool (parámetro opcional)
Respuesta	list_mangas: Object{ server_name: string manga_name: string list_chapter_readed: string[ ] (clave opcional) numchapters: int (clave opcional) imageurl: null | string (clave opcional) }[ ] next_random: string
Códigos de error	400: Los datos de sesión son incorrectos y el usuario debe hacer login de nuevo. 401: La clave generada a partir del PIN y el número aleatorio no coincide. 403: El usuario que ha iniciado la sesión no coincide con el usuario al que se quiere acceder.

Tabla 27: API REST (Obtener lista de favoritos)

Nombre	Obtener lista de capítulos leídos
Descripción	Devuelve la lista de capítulos leídos de un usuario. El manga del que se quiere obtener esta información debe estar marcado como favorito por el usuario.
Tipo	GET
Ruta	/favourite/{user}/{server}/{manga}
Parámetros	client_id: string key: string user: string (parámetro enviado mediante la ruta) server: string (parámetro enviado mediante la ruta) manga: string (parámetro enviado mediante la ruta)
Respuesta	list_chapter_readed: string[ ] next_random: string
Códigos de error	400: Los datos de sesión son incorrectos y el usuario debe hacer login de nuevo. 401: La clave generada a partir del PIN y el número aleatorio no coincide. 403: El usuario que ha iniciado la sesión no coincide con el usuario al que se quiere acceder.

Tabla 28: API REST (Obtener lista de capítulos leídos)

Nombre	Modificar lista de capítulos leídos
Descripción	Añade o quita uno o más capítulos de la lista de capítulos leídos a un favorito de un usuario. El favorito al que se le quieren añadir o quitar capítulos leídos debe existir previamente.
Tipo	POST
Ruta	/favourite/{user}/{server}/{manga}
Parámetros	client_id: string key: string user: string (parámetro enviado mediante la ruta) server: string (parámetro enviado mediante la ruta) manga: string (parámetro enviado mediante la ruta) type: string (valores admitidos: add | delete) chapter: string | string[ ]
Respuesta	next_random: string
Códigos de error	400: Los datos de sesión son incorrectos y el usuario debe hacer login de nuevo. 401: La clave generada a partir del PIN y el número aleatorio no coincide. 403: El usuario que ha iniciado la sesión no coincide con el usuario al que se quiere acceder. 500: El parámetro “type” contiene un valor no admitido.

Tabla 29: API REST (Modificar lista de capítulos leídos)

Nombre	Obtener lista de fuentes
Descripción	Devuelve la lista de fuentes (en la API llamados servidores).
Tipo	GET
Ruta	/server
Parámetros	-
Respuesta	list_servers: string[ ]
Códigos de error	-

Tabla 30: API REST (Obtener lista de fuentes)

Nombre	Obtener información de una fuente
Descripción	Devuelve la lista de mangas de la fuente especificada.
Tipo	GET
Ruta	/server/{server}
Parámetros	server: string (parámetro enviado mediante la ruta)
Respuesta	name: string url: string list_mangas: string[ ]
Códigos de error	400: No existe la fuente.

Tabla 31: API REST (Obtener información de una fuente)

Nombre	Obtener información de un manga
Descripción	Devuelve la información y la lista de capítulos de un manga, especificado mediante el nombre de la fuente a la que pertenece y el nombre del manga.
Tipo	GET
Ruta	/manga/{server}/{manga}
Parámetros	server: string (parámetro enviado mediante la ruta) manga: string (parámetro enviado mediante la ruta)
Respuesta	server_name: string name: string url_image: null | string url_server_manga: string description: string list_chapters: string[ ] ranking: int
Códigos de error	400: No existe el manga.

Tabla 32: API REST (Obtener información de un manga)

Nombre	Obtener información de un capítulo
Descripción	Devuelve la lista de imágenes de un capítulo, especificado mediante el nombre de la fuente a la que pertenece, el nombre del manga y el nombre del capítulo.
Tipo	GET
Ruta	/manga/{server}/{manga}/{chapter}
Parámetros	server: string (parámetro enviado mediante la ruta) manga: string (parámetro enviado mediante la ruta) chapter: string (parámetro enviado mediante la ruta)
Respuesta	list_images: string[ ]
Códigos de error	400: No existe el capítulo.

Tabla 33: API REST (Obtener información de un capítulo)

4.3.5. Otros

4.3.5.1. Icono de la aplicación

Ilustración 35: Icono de la aplicación

5. Implementación

5.1. Introducción

Esta sección contiene la información relativa a la implementación del cliente de escritorio y el servidor.

5.2. Cliente de escritorio

Para el desarrollo del cliente de escritorio se ha utilizado el IDE QtCreator, especialmente creado para el desarrollo de aplicaciones Qt, con todo el framework muy bien integrado y con ayudas específicas como el diseñador gráfico de interfaces Qt.

5.2.1. Comunicación con el servidor

La comunicación con el servidor se hace mediante una API REST, por llo cual con hacer conexiones HTTP es suficiente.

Para esto se ha aprovechado las clases y métodos proporcionados por la librería de red de Qt, la cual está bien integrada con el sistema de eventos de Qt y proporciona todo lo necesario para fácilmente realizar peticiones asíncronas.

A continuación un ejemplo de la petición para obtener la lista de mangas de una fuente:

```
QUrl url = ServerManager::urlServer(server_name);
QNetworkRequest request(url);
QNetworkReply *reply = qNetAccess.get(request);

connect(reply, &QNetworkReply::finished,
        this, &MangasList::onSignal_updateList_finished);
```

Código 1: Iniciar petición HTTP

En esta sección de código se puede observar cómo se crea una petición HTTP asíncrona. *ServerManager::urlServer*, a partir del nombre de una fuente, construye la url a la que hay que llamar para obtener la lista de mangas de esa fuente y la devuelve en forma de cadena de texto. *qnetAccess* simplemente es un objeto de tipo *QnetworkAccessManager* que se crea como variable de clase y gestiona las distintas conexiones y se encarga de cerrar cualquier conexión al destruir la clase, de forma que al cerrar una ventana todas las conexiones de red asociadas a esa ventana también terminen.

Tras crear la petición, se conecta la señal de finalización de la respuesta que se genera al iniciar la petición con un método dentro de la misma clase llamado *onSignal_updateList_finished*. Cuando se termine la descarga este método será llamado.

El slot que se ejecuta al terminar la petición HTTP se construye de la siguiente manera:

```
void MangasList::onSignal_updateList_finished()
{
    QNetworkReply *reply = (QNetworkReply *) sender();
    int status = reply
        ->attribute(QNetworkRequest::HttpStatusCodeAttribute).toInt();

    if(reply->error() == QNetworkReply::NoError && status == 200) {

        // Procesar datos descargados

    } else if(reply->error() != QNetworkReply::OperationCanceledError) {

        // Mostrar mensaje de error

    }

    reply->deleteLater();
}
```

Código 2: Procesar petición HTTP finalizada

Para las peticiones POST se pueden añadir los parámetros a la petición de la siguiente manera:

```
QUrl url = ServerManager::urlAddFavourite(userName, clientId, getKey());

QUrlQuery query;
query.addQueryItem("server", server);
query.addQueryItem("manga", manga);
for(const QString &chapter : chapters)
    query.addQueryItem("chapter[]", chapter);

QByteArray body;
body.append(query.toString());

QNetworkRequest request(url);
request.setHeader(QNetworkRequest::ContentTypeHeader,
                  "application/x-www-form-urlencoded");
QNetworkReply *reply = qNetAccess.post(request, query.toString().toB);

connect(reply, &QNetworkReply::finished,
        this, &ServerLogin::requestFinished);
```

Código 3: Iniciar petición HTTP POST

Para obtener el progreso de una descarga, cosa que utilizamos al descargar imágenes, para mostrar el progreso. Sólo es necesario conectar la señal adecuada. En este caso, además, nos aprovechamos de las funciones anónimas de C++11, las

cuales están soportadas por Qt, para introducir un parámetro del entorno (*numPage*) y que se utilice junto a los datos enviados en la señal para llamar al método que renderizará el progreso de la descarga.

```
connect(reply, &QNetworkReply::downloadProgress, [this, numPage](qint64
bytesReceived, qint64 bytesTotal) {
    onSignal_donloadImage_downloadProgress(numPage, bytesReceived, bytesTotal);
});
```

Código 4: Iniciar petición HTTP POST

Por último, es necesario interpretar el JSON devuelto por el servidor. Para ello utilizaremos las clases que Qt provee para tal fin. Siguiendo el primer ejemplo de descargar los mangas de una fuente, podríamos utilizar este código para extraer los nombres de los mangas y guardarlos en el objeto cache.

```
QString body = reply->readAll();
QJsonDocument jsonResponse = QJsonDocument::fromJson(body.toUtf8());
QJsonObject jsonObject = jsonResponse.object();
QJsonArray jsonArray = jsonObject["list_mangas"].toArray();

QStringList &mangasList = cache[jsonObject["name"].toString()];
for(QJsonValue jsonValue : jsonArray) {
    mangasList.append(jsonValue.toString());
}
```

Código 5: Decodificar JSON

5.2.2. Persistencia

Para guardar el estado de la aplicación entre ejecuciones se ha elegido la base de datos SQLite, para la cual Qt ofrece soporte nativo. Esta base de datos se guarda en un archivo el cual es manejado directamente por el cliente. No requiere ningún tipo de servidor a parte para que maneje la base de datos.

En la aplicación, primero creamos un directorio donde la aplicación guardará sus archivos, el cuál nos facilita Qt. Tras esto creamos el directorio si no existiera y abrimos la base de datos. Ya que las bases de datos SQLite no tienen un nombre de base de datos, en Qt el driver de SQLite aprovecha este parámetro como nombre del archivo de base de datos. Así para abrir la base de datos lo haríamos de la siguiente manera:

```
// Establecer directorio donde guardar la base de datos
QDir dbPath = QStandardPaths::writableLocation(QStandardPaths::DataLocation);
if(!dbPath.exists())
    !dbPath.mkpath(dbPath.path());

// Abrir la base de datos
db = QSqlDatabase::addDatabase("QSQLITE");
db.setDatabaseName(dbPath.path() + QDir::separator() + DB_FILE_NAME);
db.open();
```

Código 6: Abrir base de datos SQLite

Para hacer cambios en la Base de datos utilizamos se utiliza el lenguaje SQL, pero creamos la query utilizando la clase *QSqlQuery* el cual se encarga de montar la query escapando los elementos a insertar en ella y escribiéndolos en el formato adecuado a su tipo. Por ejemplo para eliminar un capítulo marcado como leído se haría de la siguiente forma:

```
QSqlQuery query(db);
query.prepare(QStringLiteral("DELETE FROM readed_chapters WHERE server=? AND
manga=? AND chapter=?"));
query.bindValue(0, server_name);
query.bindValue(1, manga_name);
query.bindValue(2, chapter_name);
query.exec();
```

Código 7: Ejecutar una query con QSqlQuery

Esto internamente utiliza los *prepared staments* de SQL, lo cual nos permite crear una query y utilizarla varias veces con distintos parámetros. Así, para marcar como leídos varios capítulos de un manga se puede hacer de esta manera:o

```
QSqlQuery query(db);
query.prepare("INSERT INTO readed_chapters (server, manga, chapter) VALUES (?, ?,
?)");
query.bindValue(0, server_name);
query.bindValue(1, manga_name);

for(const QString &chapter_name : chapters) {
    query.bindValue(2, chapter_name);
    query.exec();
}
```

Código 8: Ejecutar varias queries con QSqlQuery

En un principio la base de datos era bastante lenta. Si al hacer login e importar los datos de servidor, tenía que marcar como leídos 500 capítulos, podía tardar 50 segundos esta operación (100 ms por query). Por ello se cambiaron algunos parámetros de la base de datos. Para ello se añadieron unas sentencias SQL que se ejecutan tras crear la base de datos y establece unas propiedades en el driver.

```
// Reducir los accesos a disco para mejorar el rendimiento de la base de datos
QSqlQuery query(db);
query.exec("PRAGMA page_size = 4096");
query.exec("PRAGMA cache_size = 16384");
query.exec("PRAGMA temp_store = MEMORY");
query.exec("PRAGMA journal_mode = OFF");
query.exec("PRAGMA locking_mode = EXCLUSIVE");
query.exec("PRAGMA synchronous = OFF");
```

Código 9: Optimizar rendimiento de SQLite

Tras hacer estos cambios, el tiempo de marcar 500 mangas como leídos se redujo a 0.5 segundos (1 ms por query). Esto significa que la base de datos se volvió 100 veces más rápida ejecutando operaciones de modificación de filas. En lectura la mejora es mucho menos notable, ya que desde el principio ya tenía buen rendimiento en estas operaciones.

5.2.3. Sincronización de favoritos.

Para la sincronización de favoritos, el servidor simplemente provee la lista de favoritos que tiene guardada, y es el cliente el que se encarga de mezclar estos datos con los que tiene en local.

Un usuario puede iniciar sesión en varios clientes y estos deben mantener los datos sincronizados. Por otra parte, si un usuario inicia sesión por primera vez, pero ya tenía mangas en favoritos con capítulos leídos, estos deben guardarse. Por esto se ha desarrollado este algoritmo.

Tras iniciar sesión, si un manga ya estaba en favoritos según el servidor, sobrescribe los datos de capítulos leídos de este manga con los del servidor. En cambio, si no existía, lo añade.

Mientras tanto, si ya se hizo la primera sincronización, y simplemente está descargando los datos tras el inicio de la aplicación para ver si hay cambios, supone que cualquier cambio en el servidor ha sido hecho por otro cliente y da prioridad absoluta a los datos del servidor. Así que el cliente modifica sus datos de favoritos y sus capítulos leídos para que queden igual que en el servidor.

```
// Procesar los favoritos que existen en el servidor
QSet<Favourite> serverFavourites;
for (QJsonValue mangaValue : jsonObject["list_mangas"].toArray()) {
    QJsonObject mangaObj = mangaValue.toObject();
    QString server = mangaObj["server_name"].toString();
    QString manga = mangaObj["manga_name"].toString();
    QString imgurl = mangaObj["imgurl"].toString();
    int numchapters = mangaObj["numchapters"].toInt();

    // Marcar como favorito del servidor
    serverFavourites.insert({server, manga, QString(), 0, 0});

    // Añadir o actualizar favorito
    if (!storage->isFavourite(server, manga))
        storage->addFavourite(server, manga, imgurl, numchapters, false);
    else
        storage->updateFavourite(server, manga, imgurl, numchapters);

    // Obtener listas de leidos del favorito
    QSet<QString> localChapters = QSet<QString>::fromList(
                storage->getReaded(server, manga));
    QSet<QString> serverChapters;
    for (QJsonValue chapterValue : mangaObj["list_chapter_readed"].toArray())
        serverChapters.insert(chapterValue.toString());

    // Borrar capítulos que no existen en el servidor
    QStringList chaptersToDelete = (localChapters-serverChapters).toList();
    storage->setReadStatus(server, manga, chaptersToDelete, false, false);

    // Añadir capítulos que no existen en local
    QStringList chaptersToAdd = (serverChapters-localChapters).toList();
    storage->setReadStatus(server, manga, chaptersToAdd, true, false);
}

// Procesar favoritos locales que no existen en el servidor
QSet<Favourite> localFavourites = storage->getFavourites();
for (Favourite fav : localFavourites - serverFavourites) {
    QString server = fav.serverName;
    QString manga = fav.mangaName;

    if (firstSync)
        // Añadir al servidor
        sendAddFavourite(server, manga, storage->getReaded(server, manga));
    else
        // Borrar de local
        storage->deleteFavourite(server, manga, false);
}

firstSync = false;
```

Código 10: Sincronización de favoritos

5.2.4. Capturas de pantalla

A continuación algunas capturas de pantalla de la aplicación en funcionamiento.

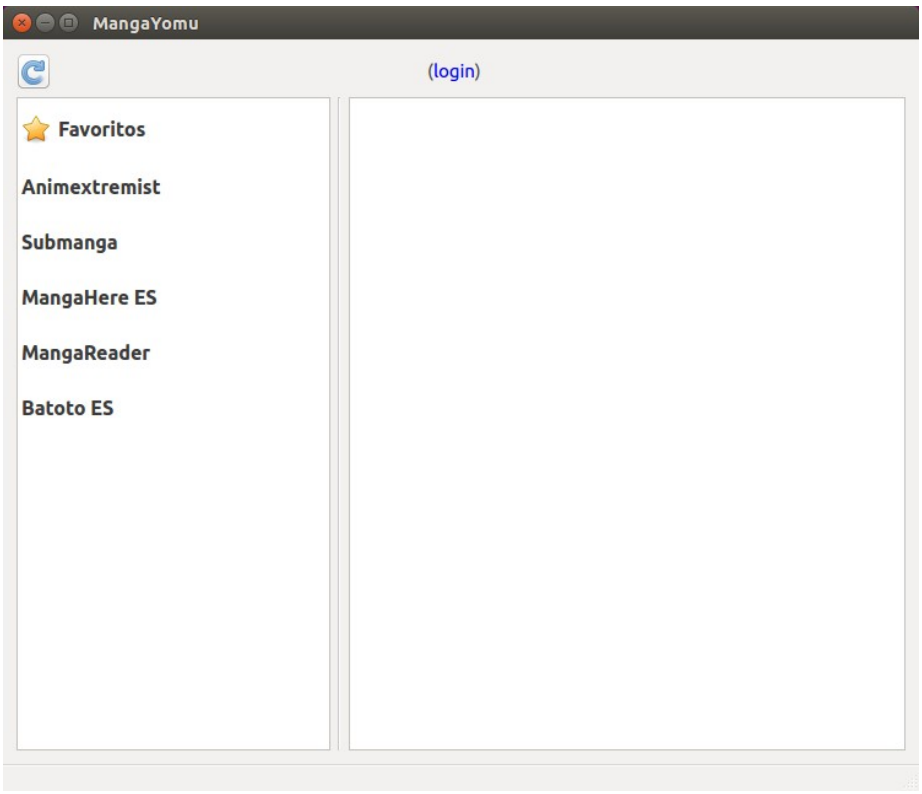

Ilustración 36: Captura de pantalla (Pantalla inicial)

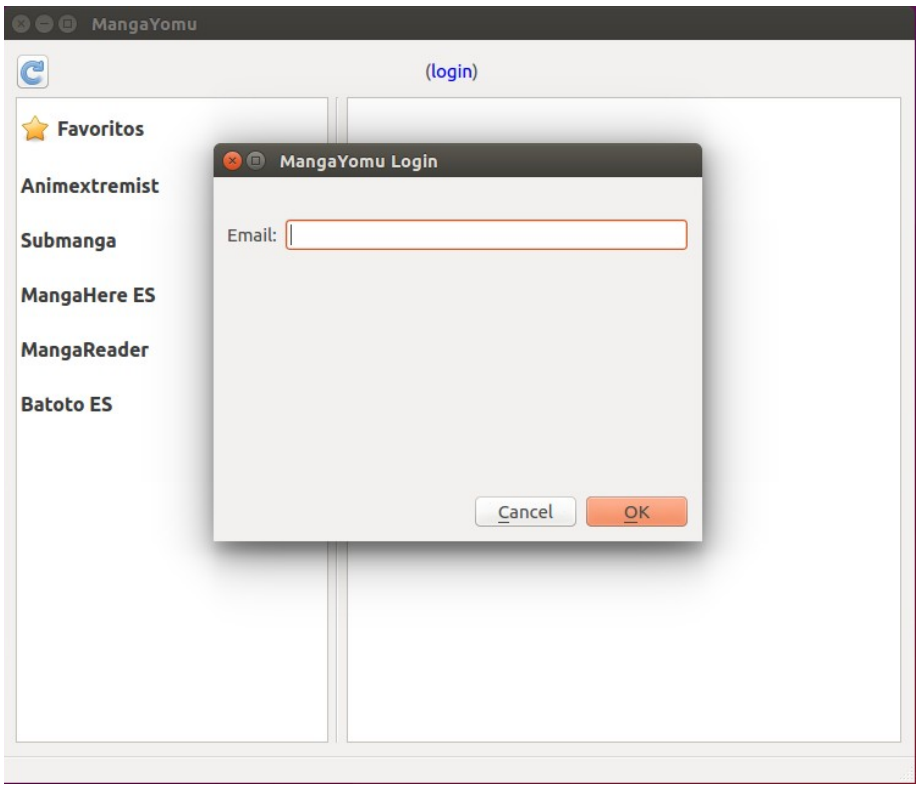

Ilustración 37: Captura de pantalla (Inicio de sesión: Introducir email)

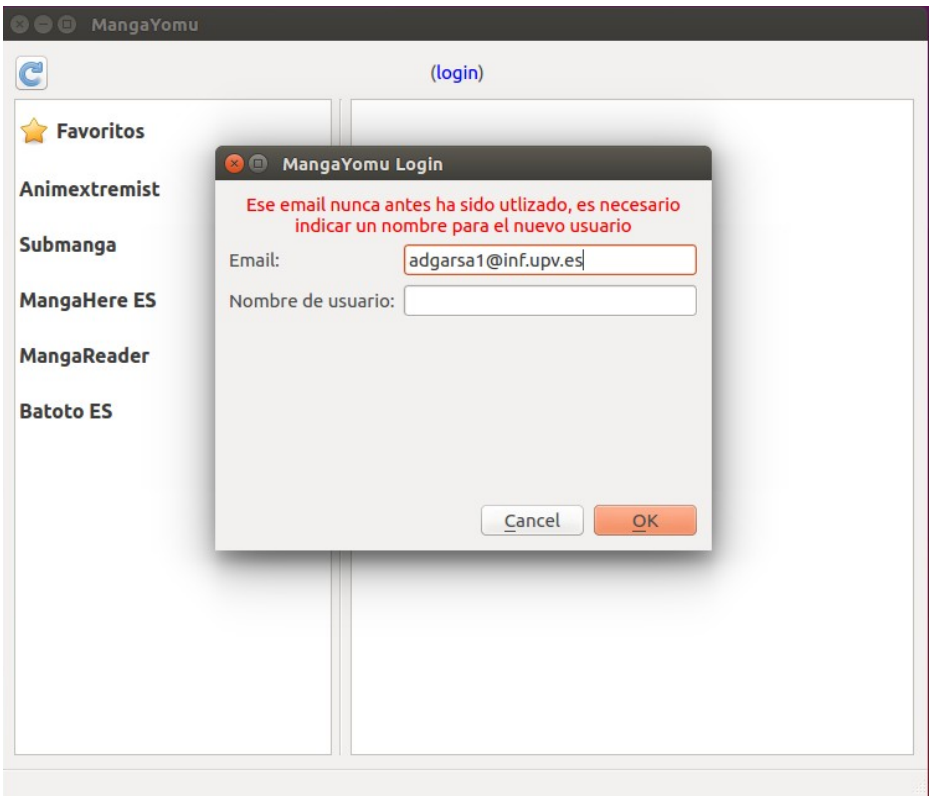

Ilustración 38: Captura de pantalla (Inicio de sesión: Introducir nombre de usuario)

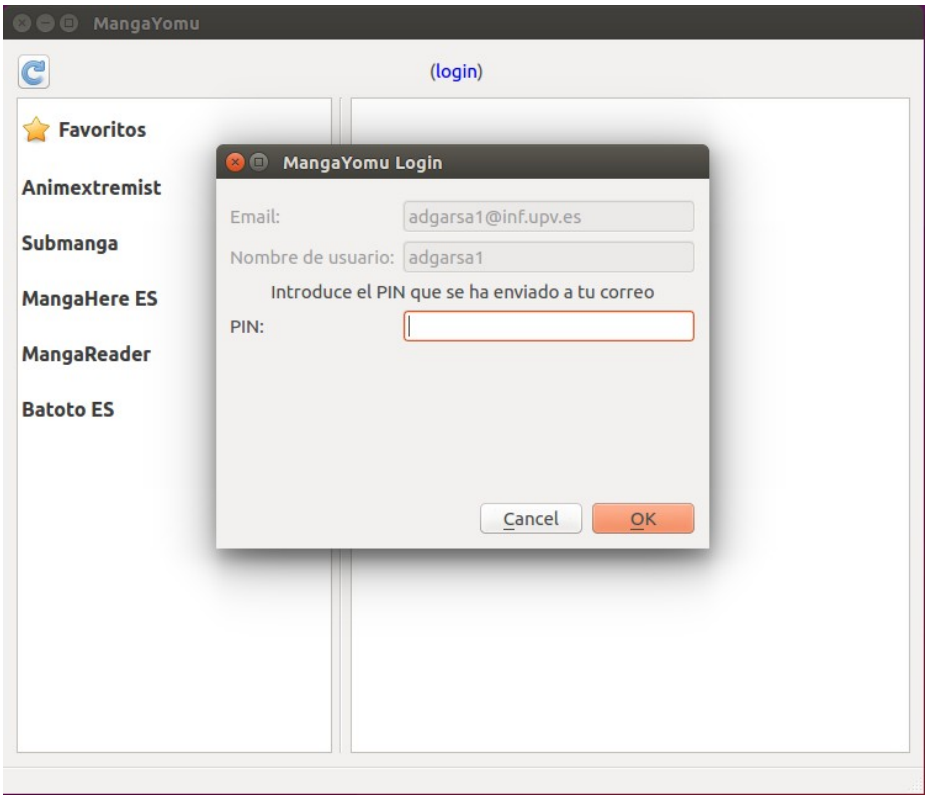

Ilustración 39: Captura de pantalla (Inicio de sesión: Introducir PIN)

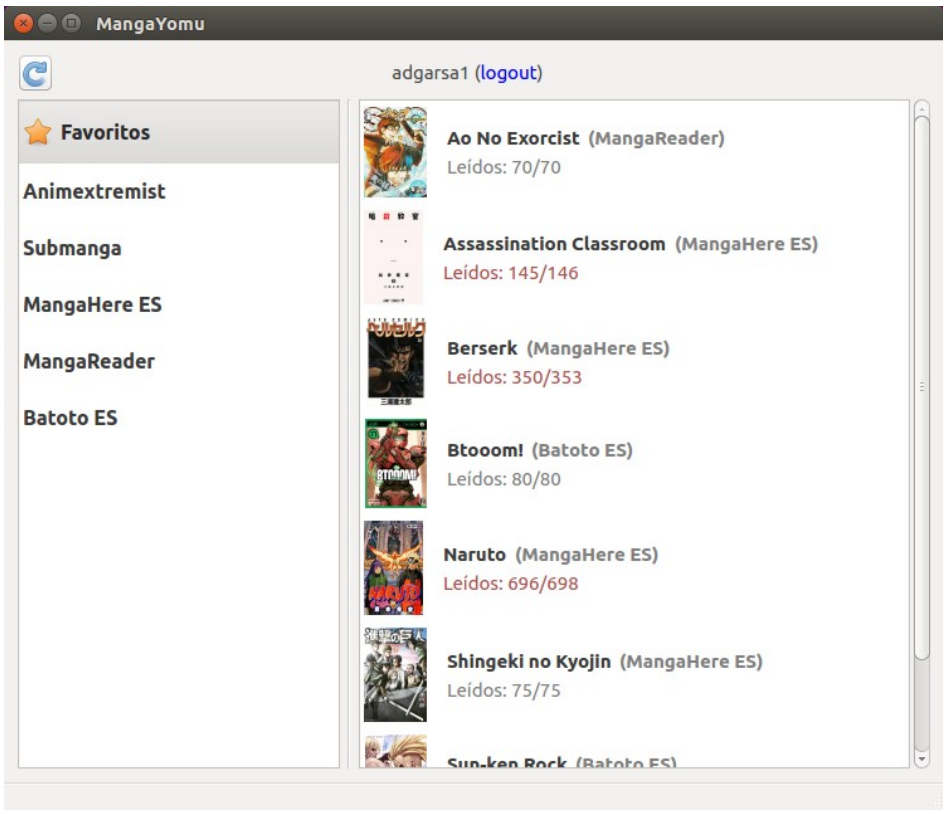

Ilustración 40: Captura de pantalla (Favoritos)

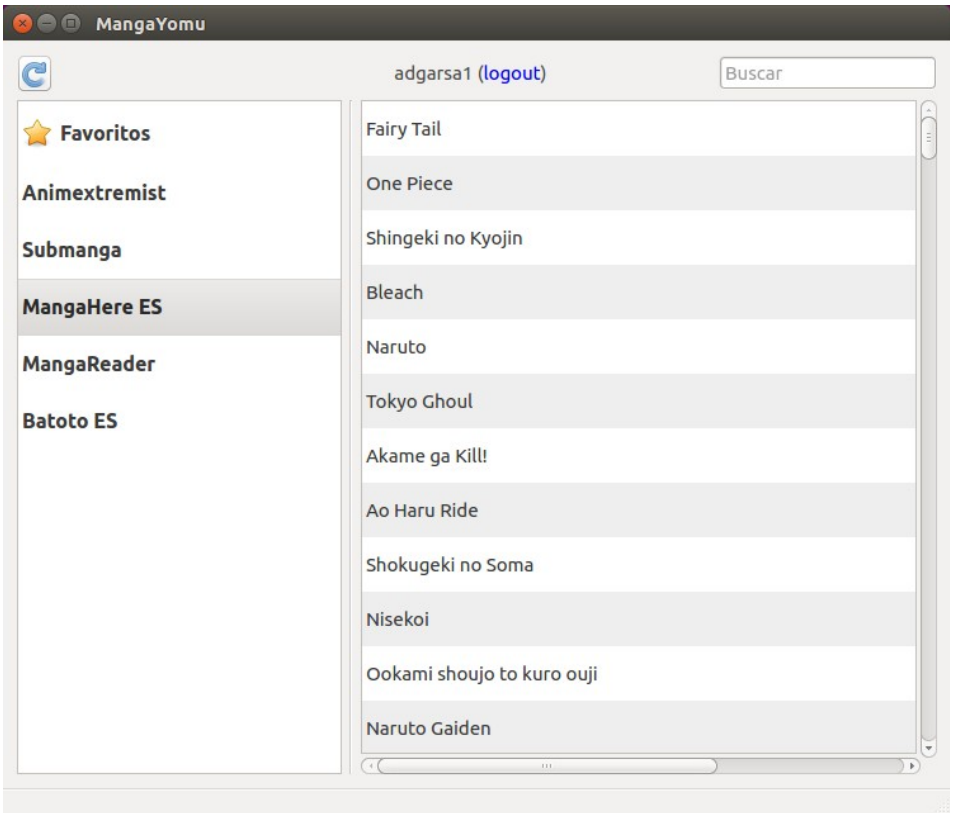

Ilustración 41: Captura de pantalla (Mangas de una fuente)

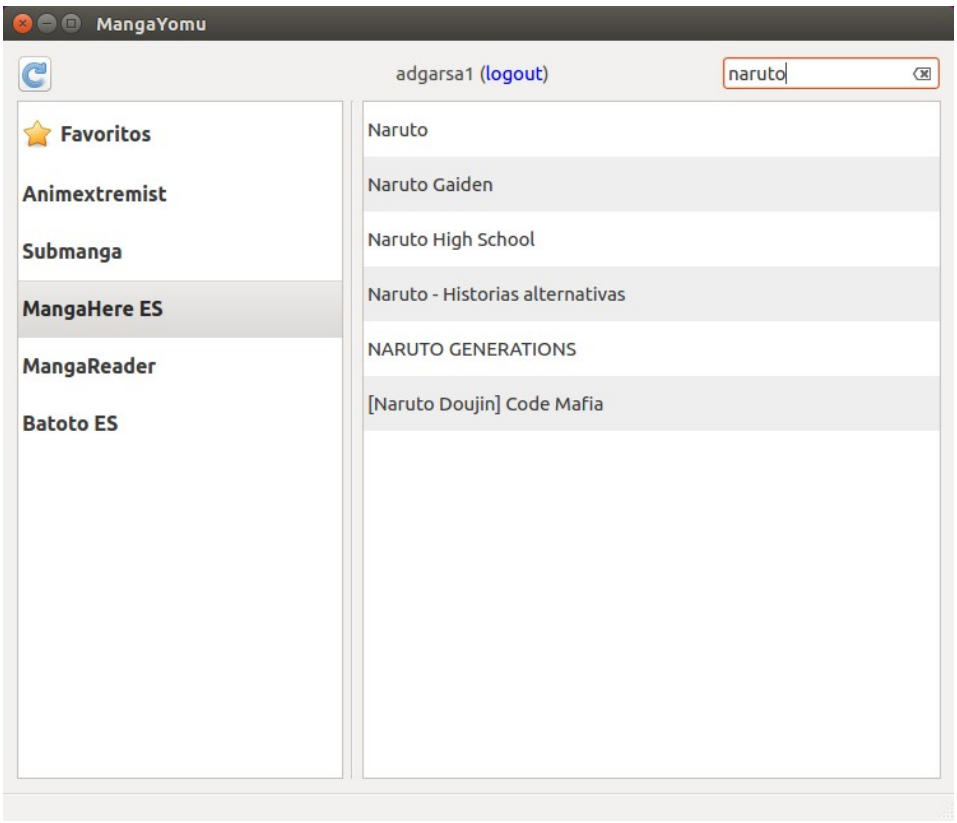

Ilustración 42: Captura de pantalla (Búscar en la lista de mangas)

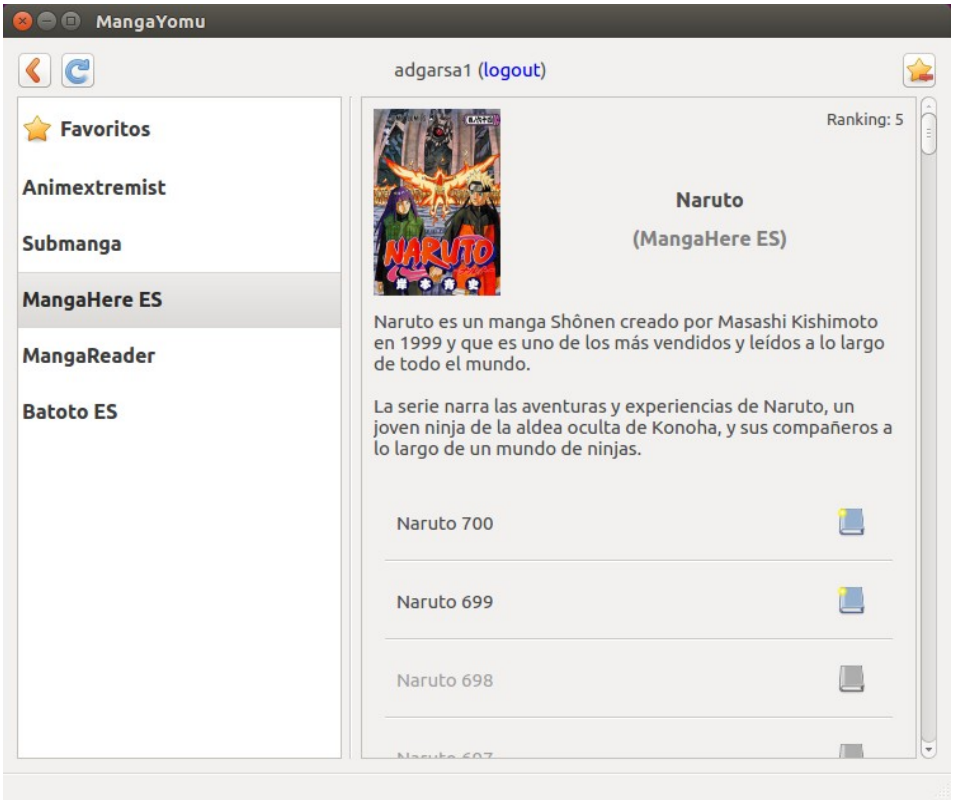

Ilustración 43: Captura de pantalla (Información y lista de capítulos de un manga)

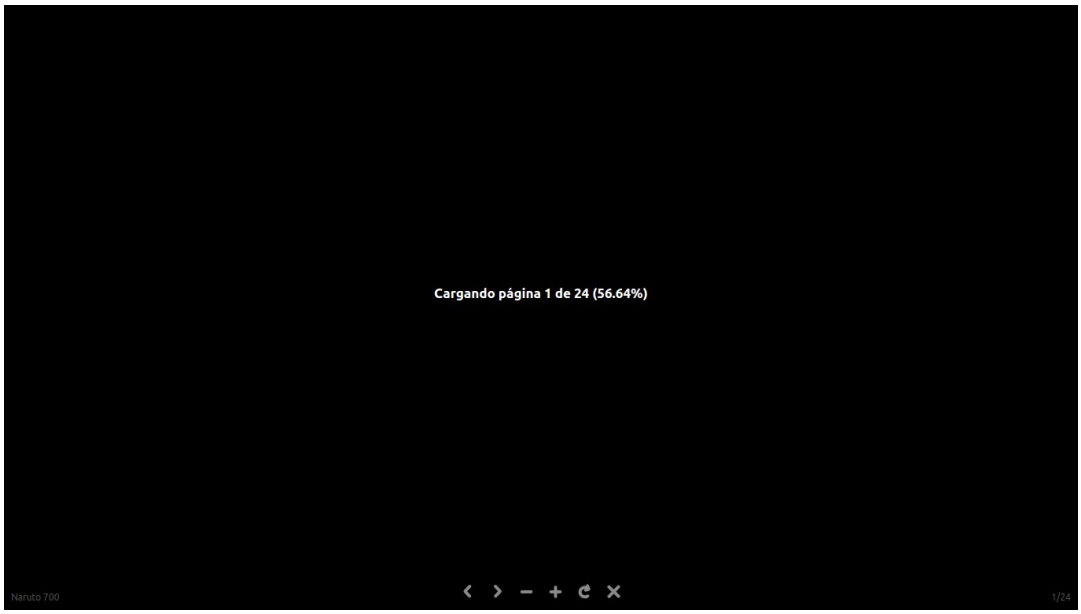

Ilustración 44: Captura de pantalla (Visor de capítulo: Cargando página)

Ilustración 45: Captura de pantalla (Visor de capítulo: Inicio de la página)

Ilustración 46: Captura de pantalla (Visor de capítulo: Final de la página)

Ilustración 47: Captura de pantalla (Visor de capítulo: Final del capítulo)

5.3. Servidor

5.3.1. Servicio REST

Para el servicio rest es necesario exponer un servicio HTTP que escuche peticiones. En este caso se ha utilizado Sinatra, un framework de Ruby que

permite definir fácilmente servicios HTTP, indicando qué debe devolver en cada ruta. Su sintaxis sencilla y visual facilita mucho crear interfaces REST.

Como se puede ver en el siguiente ejemplo crear un servidor sencillo con una ruta a la que responde con un mensaje predefinido no cuesta más de 4 lineas.

```
require 'sinatra'

get '/hi' do
  "Hello World!"
end
```

Código 11: Servidor básico con Sinatra

Un ejemplo más completo de lo que ofrece Sinatra podría ser la implementación del método para obtener la lista de mangas de una fuente. En este caso, primero busca los datos de la fuente en la base de datos. Si no la encuentra devuelve el código de error 400. Si encuentra los datos, devuelve en formato JSON la información de la fuente junto con la lista de mangas de la fuente.

```
get '/server/:server' do |server|
    doc = settings.mongo_db['server'].find_one({name: server})

    if doc.nil?
        400 # No existe ese servidor
    else
        {
            name:         doc['name'],
            url:          doc['url'],
            list_mangas: doc['list_mangas']
        }.to_json
    end
end
```

Código 12: Obtener la lista de mangas de una fuente con Sinatra

5.3.2. Persistencia

Para guardar los datos se ha elegido MongoDB. Es rápida, sencilla, flexible y escalable. Guarda los datos en colecciones de documentos en formato BSON, que es una forma binaria de codificar documentos JSON. Esto hace que la forma de los datos de la base de datos sea muy cercana a la manejada por el servicio REST.

Estos documentos no tienen un formato concreto, sino que pueden contener estructuras complejas de datos, lo cual permite almacenar los datos de forma que el servidor sólo deba buscar un documento y devolver el contenido, aunque lo que deba devolver sean estructuras complejas.

Estas características se ven claras en el ejemplo de devolver la información de una fuente con Sinatra, donde simplemente busca el documento

correspondiente a la fuente y devuelve su contenido. En ese documento viene el nombre de la fuente y la lista de mangas de esa fuente. No ha necesitado hacer ningún tipo de búsqueda en la colección de mangas ni ningún agregado. Esto aumenta la eficiencia, ya que una acción tan común como obtener los mangas de una fuente la hemos reducido a leer un documento que lo buscamos a través de un campo indexado y devolver su contenido tal cual, sin necesidad de hacer búsquedas extra en otras tablas o cruzar varias tablas.

En el siguiente ejemplo se muestra como se crea la conexión a la base de datos y se actualiza un elemento en la colección de capítulos. Para ello primero busca el documento en la base de datos, si no lo encuentra usa uno nuevo. En este documento guarda la nueva lista de capítulos. Tras eso guarda el documento en la base de datos. Si encontró el documento en la base de datos, este tendrá un campo '_id' el cual es un identificador único del objeto, lo cual, al guardarlo, provocará que MondoDB sobrescriba el documento que había en la colección con ese id. En caso contrario al documento no le hemos añadido ningún id, por lo que MongoDB generará un id nuevo que no esté usado y se lo asignará, y añadirá el documento a la colección.

```
connection = Mongo::MongoClient.new
db = connection.db 'mangayomu'

doc = {
    'server_name' => server,
    'manga_name'  => manga,
    'name'        => chapter
}
db_doc = db['chapter'].find_one(doc) || doc

db_doc['list_images'] = list_images
db['chapter'].save db_doc
```

Código 13: Actualizar documento en MongoDB

5.3.3. Escaneador de fuentes

Para extraer la información de mangas de las distintas fuentes se ha ha creado un sistema de proveedores de información.

Primero se ha creado una clase base llamada *Provider* que implementa todas las funciones básicas para buscar información en los mangas. Por otra parte se ha implementado un método que recoge la información de la fuente y la guarda en la base de datos MondoDB de la que luego lee los datos el servidor REST.

Para que el método que lee los datos de la fuente funcione es necesario para cada fuente crear una clase que extienda a *Provider* y defina unas constantes y

unos métodos concretos que recojan información para el método que escanea toda la fuente. De esta forma todo el código común queda en la clase *Provider* y luego sólo es necesario crear una pequeña clase con unos pocos métodos para cada fuente que implementa extraer la información de un elemento concreto de esa fuente.

Cada escaneador de fuente tiene una estructura como la siguiente:

```
#!/usr/bin/env ruby
# coding: utf-8
require_relative 'yomu-utils'

class MyProvider < Provider
	LOG_OUTPUT = 'MyProvider.log'
	SERVER_NAME = 'MyProviderName'
	URL = 'http://my.provider.url'
	MANGAS_URL = "#{URL}/path/to/all/mangas"

	def get_mangas url
		# Devolver una lista de urls de los mangas que contiene la página
	end

	def get_info_and_capitulos url
		# TODO un único Hash con la información relativa al manga
	end

	def get_imagenes_paginas url
		# TODO una lista con las urls de las imágnes de cada página
		 # del capítulo
	end
end


if __FILE__ == $0
	provider = MyProvider.new
	fast = ARGV.include?('--complete') ? false : true
	provider.update_provider fast
end
```

Código 14: Escaneador de una fuente

Al iniciar el servidor, este lanza un trabajo paralelo que importa todas estas clases y una vez al día ejecuta el método *update_provider* de cada una.

También es posible ejecutar el archivo directamente desde la linea de comandos y esto comenzará el proceso de escaneo de la fuente si es necesario alguna operación manual.

Por defecto lanza una actualización rápida, la cual no escanea el contenido de los capítulos si estos ya se encuentran en la base de datos. Pero si es necesario, añadiendo el parámetro *--complete* al lanzarlo desde la linea de comandos, es posible iniciar una actualización completa de la fuente, la cual escanea también el contenido de todos los capítulos, actualizándolos con los nuevos datos. Además, gracias a la flexibilidad de MongoDB y al diseño elegido para la base de datos, esta

ejecución completa, la cual puede durar hasta 1 semana si la fuente tiene muchos mangas, es posible ejecutarla mientras el parser rápido sigue ejecutándose diariamente sin que se provoquen ningún tipo de interferencia entre si. De esta forma si las urls de imágenes de muchas páginas cambiaran es posible lanzar un trabajo extra manual que analice de nuevo todos los datos de todo el servidor para importar esto.

6. Conclusiones

6.1. Introducción

En este apartado se analizan las dificultades encontradas en la realización del proyecto y las aportaciones obtenidas. Además se proponen una serie de ampliaciones realizables a partir del resultado del proyecto.

6.2. Dificultades encontradas

Durante el desarrollo se ha notado que las operaciones de modificación de filas en la base de datos SQLite del cliente eran excesivamente lentas, mientras que las de lectura podían mejorar un poco. Para solucionar esto se añadieron unas directivas que modifican las propiedades de la cache para que los datos permanezcan más en ella, aumentando con ello el rendimiento al leer datos. Además se han desactivado el journaling y la sincronización de escritura, dos elementos que evitan la corrupción del archivo de base de datos si el ordenador se apagara bruscamente durante una escritura en el archivo, pero que hoy en día no son necesarios, ya que los propios sistemas operativos suelen implementar en sus sistemas de ficheros journaling además de procesos de recuperación de archivos ante un fallo de este tipo.

También se ha encontrado que los objetos de tipo lista que ofrece Qt no ofrecen suficiente flexibilidad a la hora de integrarse con otros elementos dentro de una zona desplazable sin mostrar sus propios desplazadores. A parte, también dificultan el uso de separadores personalizados y limitan la personalización de los elementos elementos de la lista. Estos problemas se encontraron al intentar crear la lista de capítulos de un manga tal y como se había diseñado. Es por esto que se decidió construir esta lista a base de objetos básicos como separadores y botones dispuestos en un layout vertical. Aplicando algunas propiedades CSS, lo cual Qt soporta, a algunos elementos se consiguió darle el aspecto deseado a la lista.

Descargar la lista de mangas de una fuente tarda unos cuantos segundos. Es bastante molesto para el usuario que cada vez que cambia de una fuente a otra, una acción bastante habitual, tenga que volver a esperar unos segundos mientras descarga la lista de mangas de la fuente. Para solventar esto se implementó una cache en memoria que guarda la lista de mangas de cada fuente cuando se descarga y la guarda hasta el cierre de la aplicación. De esta forma que las

siguientes veces que se vuelve a acceder a esa fuente, en vez de descargar otra vez la lista, usa la que tiene en memoria. También se ha implementado una cache de imágenes de página, pero en este caso en disco, ya que las imágenes ocupan más. Con esto, de una ejecución a otra, si un capítulo se abrió hace poco, muestra sus imágenes directamente sin necesidad de volver a descargarlas, aunque se haya cerrado el programa. Para esto se ha aprovechado una clase para gestionar este tipo de cache que ofrece Qt, la cual se integra con el gestor de descargas de Qt y aplica esta cache de forma transparente. Además, esta clase establece un limite de tamaño de la cache guardando cosas hasta que se alcanza, y borrando lo más antiguo para dejar sitio cuando este límite es alcanzado.

6.3. Aportaciones obtenidas

Se ha desarrollado la especificación de requisitos con formato IEEE 830-1998, para una aplicación distribuida lectora de mangas sobre escritorio.

Se ha realizado el desarrollo de la aplicación de escritorio multiplataforma utilizando el framework Qt con la ayuda del IDE QtCreator. También se ha diseñado y construido una interfaz de usuario con este entorno. Este framework es cada vez más completo y cada vez va siendo utilizado por más empresas para el desarrollo de sus aplicaciones. No sólo permite el desarrollo de aplicaciones de escritorio, sino también permite exportar estas aplicaciones a sistemas operativos móviles como Android o iOS, además de poder ser usado incluso para aplicaciones de servidor.

Se ha utilizado Sinatra sobre Ruby para crear un servicio REST. Éste es un framework sencillo y potente para este tipo de servicios que puede ser de mucha utilidad. También se ha integrado la utilización de una base de datos de tipo NoSQL como MongoDB, cuyo uso está aumentando con la necesidad de crear aplicaciones potentes de forma más rápida y con más flexibilidad.

6.4. Ampliaciones futuras

- **Notificaciones del servidor**: El servidor notificará en tiempo real a los clientes de cualquier cambio en las propiedades del usuario o de nuevos capítulos en los mangas favoritos de los usuarios.
- **Más opciones de registro**: Permitir el registro con redes sociales además del email.

- **Lista de amigos**: Los usuarios podrán añadir otros usuarios como amigos y podrán ver lo que han leído éstos y comparar con su propia lista de lectura, además de poder ver otras posibles opciones de usuario que se implementen en el futuro.
- **Valoración y comentarios**: Permitir valorar mangas o capítulos, además de añadir comentarios a éstos, los cuales serán públicos y podrá leer cualquier persona.
- **Mejorar uso del programa sin acceso a internet**: Ampliar el uso de la cache de disco para que un usuario pueda encender el programa y volver a abrir un manga al que accedió hace poco aunque no se tenga acceso a internet. También sería útil para este fin, añadir junto a cada capítulo un botón que descargue todas las imágenes del capítulo sin necesidad de entrar en él. Por último, se debe añadir un sistema de persistencia para las peticiones de modificación de datos de usuarios registrados de forma que aunque no haya conexión a internet sea posible cerrar la aplicación, y que al volverla a abrir termine las acciones pendientes.

7. Bibliografía

Five Best Desktop Comic Book Readers, http://lifehacker.com/5858906/five-best-desktop-comic-book-readers [24 de Septiembre de 2015]

How To Read Comic Books On Your Computer, http://www.howtogeek.com/80091/how-to-read-comic-books-on-your-computer-2/ [24 de Septiembre de 2015]

Desktop Manga Reader To Download/Read Manga from OneManga and more, http://www.technorms.com/2050/download-and-read-your-favorite-manga-on-desktop-with-comic-shelf [24 de Septiembre de 2015]

CDisplay Alternatives and Similar Software, http://alternativeto.net/software/cdisplay-comic-reader/ [24 de Septiembre de 2015]

CDisplay – The Official Site, http://www.cdisplay.me/ [24 de Septiembre de 2015]

GonVisor: El mejor programa lector de cómics para PC (CBR), http://www.gonvisor.com/es/ [24 de Septiembre de 2015]

ComicRack – The best Comic Reader in the World, http://comicrack.cyolito.com/ [24 de Septiembre de 2015]

HaruNeko download, http://sourceforge.net/projects/hakuneko/ [24 de Septiembre de 2015]

ComicShelf, http://www.comicshelf.com/en/index.php [24 de Septiembre de 2015]

Representational State Transfer – Wikipedia, https://es.wikipedia.org/wiki/Representational_State_Transfer [17 de Septiembre de 2015]

JSON – Wikipedia, https://es.wikipedia.org/wiki/JSON [17 de Septiembre de 2015]

Qt Documentation, http://doc.qt.io/ [17 de Septiembre de 2015]

Sinatra: Documentation, http://www.sinatrarb.com/documentation.html [17 de Septiembre de 2015]

The MongoDB 3.0 Manual, http://docs.mongodb.org/manual/ [17 de Septiembre de 2015]

Pragma staments supported by SQLite, https://www.sqlite.org/pragma.html [17 de Septiembre de 2015]