

UNIVERSIDAD REY JUAN CARLOS

DOBLE LICENCIATURA EN INGENIERÍA SUPERIOR EN INFORMÁTICA
Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS



PROYECTO FIN DE CARRERA

**BIKESMANAGER: SISTEMA CLIENTE/SERVIDOR
PARA LA GESTIÓN DE PARQUES DE BICICLETAS**

AUTOR: LÓPEZ CEREZO, ALEJANDRO M.

TUTOR: FERNÁNDEZ GIL, ALBERTO

Agradecimientos

A mi tutor, Alberto Fernández, por haberme dado la oportunidad de desarrollar el proyecto bajo su supervisión cuando, después de una desafortunada experiencia previa, me quedé sin ninguno asignado.

A mis padres, mi hermano y mi novia, Ana, por su comprensión y apoyo constante durante el proceso y, sobre todo, en esos fines de semana de dedicación interminables en los que no estaba para nada ni nadie después de toda la semana de trabajo.

A todos, gracias.

Resumen

El presente proyecto ha supuesto la elaboración de una aplicación Android distribuida para la gestión de parques de bicicletas, estableciendo una infraestructura tecnológica base para desarrollos e investigaciones futuras.

La aplicación permite simular las acciones de coger, dejar y reservar bicicletas y anclajes entre un conjunto de estaciones. Se introduce, además, un primer nivel de adaptación dinámica de precios acorde a la disponibilidad de cada estación, de manera que coger una bicicleta de una estación con baja disponibilidad supone una tarifa mayor que en otro puesto.

La herramienta presenta un mapa actualizado de las estaciones disponibles sobre el que poder operar, así como pantallas adicionales para facilitar la interacción del usuario con la aplicación a la hora de gestionar su perfil o conocer el estado del parque de bicicletas.

Desde un punto de vista técnico, la aplicación sigue una arquitectura Cliente/Servidor de tres capas, estructurándose la comunicación mediante el estándar REST de los servicios web.

El desarrollo ha seguido un modelo de proceso incremental bajo un paradigma orientado a objetos, tratando de tener en todo momento en mente las buenas prácticas establecidas por la Ingeniería del Software, las *Reglas de Oro* para el diseño de interfaces de usuario, recomendaciones de diseño e implementación Android, etc. Dado el modelo señalado, el proceso se ha dividido en una serie de incrementos jerarquizados por importancia y desarrollados de manera individual para, finalmente, quedar integrados en el producto final.

La aplicación ha sido probada mediante pruebas planificadas de unidad, de integración y de validación para tratar de lograr una elevada cobertura de errores y calidad final.

Se espera, como se ha indicado, que la herramienta presentada suponga una infraestructura base consistente para investigaciones futuras que conduzcan a modelos de gestión más eficientes y eficaces.

Índice general

Agradecimientos	III
Resumen	v
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Metodología	2
2. Descripción informática	5
2.1. Especificación de Requisitos Software	5
2.1.1. Introducción	5
2.1.2. Descripción General	6
2.1.3. Requisitos Específicos	8
2.2. Análisis	25
2.2.1. Introducción	25
2.2.2. Análisis conceptual del sistema	26
2.2.3. Análisis y desarrollo de requisitos	28
2.3. Diseño	40
2.3.1. Introducción	40
2.3.2. Diseño de la arquitectura	40
2.3.3. Diseño de las interfaces	46
2.4. Implementación	51
2.4.1. Introducción	51
2.4.2. Herramientas utilizadas	51
2.4.3. Distribución física	55
3. Pruebas	59
3.1. Introducción	59
3.2. Estrategia de pruebas	60
3.2.1. Pruebas de unidad	61

3.2.2. Pruebas de integración	64
3.2.3. Pruebas de validación	64
3.2.4. Pruebas de sistema	72
4. Conclusiones	73
4.1. Líneas futuras	74
A. API implementada	77
B. Resultados de las pruebas de validación	83
C. Manual de usuario	87
D. Manual de instalación	99

Índice de figuras

1.1. El modelo incremental	3
2.1. Diagrama de casos de uso	7
2.2. Diagrama de actividad de RF01: Registrar usuario	10
2.3. Diagrama de actividad de RF02: Loguear usuario	11
2.4. Diagrama de actividad de RF03: Modificar perfil	12
2.5. Diagrama de actividad de RF04: Borrar perfil	13
2.6. Diagrama de actividad de RF05: Ingresar saldo	14
2.7. Diagrama de actividad de RF06: Coger bicicleta	16
2.8. Diagrama de actividad de RF07: Dejar bicicleta	17
2.9. Diagrama de actividad de RF08: Reservar	19
2.10. Diagrama de actividad de RF09: Cancelar reserva (explícita)	20
2.11. Diagrama de actividad de RF10: Pagar	21
2.12. Diagrama de actividad de RF11: Actualizar	22
2.13. El modelo de análisis como puente entre los requerimientos y el diseño del software	26
2.14. Diagrama de clases de análisis	28
2.15. Diagrama de colaboración del camino principal del RF01	29
2.16. Diagrama de colaboración del primer camino alternativo del RF01 .	29
2.17. Diagrama de colaboración del segundo camino alternativo del RF01	30
2.18. Diagrama de colaboración del camino principal del RF02	30
2.19. Diagrama de colaboración del primer camino alternativo del RF02 .	31
2.20. Diagrama de colaboración del segundo camino alternativo del RF02	31
2.21. Diagrama de colaboración del camino principal del RF03	32
2.22. Diagrama de colaboración del primer camino alternativo del RF03 .	32
2.23. Diagrama de colaboración del segundo camino alternativo del RF03	32
2.24. Diagrama de colaboración del camino principal del RF04	33
2.25. Diagrama de colaboración del primer camino alternativo del RF04 .	33
2.26. Diagrama de colaboración del camino principal del RF05	34
2.27. Diagrama de colaboración del camino principal del RF06	34
2.28. Diagrama de colaboración del primer camino alternativo del RF06 .	35

2.29. Diagrama de colaboración del camino principal del RF07	35
2.30. Diagrama de colaboración del primer camino alternativo del RF07 .	36
2.31. Diagrama de colaboración del camino principal del RF08	37
2.32. Diagrama de colaboración del primer camino alternativo del RF08 .	37
2.33. Diagrama de colaboración del camino principal del RF09	38
2.34. Diagrama de colaboración del primer camino alternativo del RF09 .	38
2.35. Diagrama de colaboración del camino principal del RF10	39
2.36. Diagrama de colaboración del camino principal del RF11	39
2.37. Diagrama de colaboración del primer camino alternativo del RF11 .	40
2.38. Esquema de la arquitectura Cliente/Servidor de tres capas	41
2.39. Diagrama de clases de diseño de la capa del cliente	44
2.40. Diagrama de clases de diseño de la capa del servidor	45
2.41. Fragmentos en el diseño de IUs	49
2.42. Proceso de diseño y construcción de una IU	49
2.43. Relación de interfaces de usuario y requisitos funcionales	50
2.44. Datos oficiales Android de distribución de versiones a diciembre de 2016	53
2.45. Diagrama de despliegue del sistema	56
3.1. Pasos de las pruebas software	60
3.2. Ejemplo de un toast en Android	62
3.3. RESTClient, depurador de servicios web RESTful	63
C.1. Configuración inicial de la conexión	87
C.2. Pantalla inicial de la aplicación	88
C.3. Registro de nuevos usuarios	88
C.4. Inicio de sesión	89
C.5. Pantalla principal de la aplicación	90
C.6. Detalle de una estación	90
C.7. Diferentes vistas y opciones	91
C.8. Listado detallado del conjunto de estaciones	91
C.9. Estado global del parque de estaciones	92
C.10. Menú desplegable y opciones adicionales	92
C.11. Opciones a la hora de operar con una estación	93
C.12. Actualización de la barra de estado al coger una bici	94
C.13. Reserva de bicicleta y anclajes	94
C.14. Cuenta de usuario	95
C.15. Configuraciones posibles	96
C.16. “Superusuario” activado	96
D.1. Pantalla principal de MySQL Installer	100

D.2. Instalación de MySQL Server	100
D.3. Instalación de MySQL Server	101
D.4. Configuración de la conexión en MySQL Workbench	102
D.5. Ejecución de consultas SQL	103
D.6. Carga de datos en MySQL Workbench	103
D.7. Conexión del Connector/J	104
D.8. Configuración de la conexión	105
D.9. Vista de la base de datos desde NetBeans	105
D.10. Conexión base de datos	105
D.11. Conexión base de datos	106
D.12. Carga del proyecto	106
D.13. Configuración de claves en “glassfish-resoruces.xml”	107
D.14. Encendiendo GlassFish	107

Índice de tablas

2.1. Descripción textual de RF01: Registrar usuario	9
2.2. Descripción textual de RF02: Loguear usuario	11
2.3. Descripción textual de RF03: Modificar perfil	12
2.4. Descripción textual de RF04: Borrar perfil	13
2.5. Descripción textual de RF05: Ingresar saldo	14
2.6. Descripción textual de RF06: Coger bicicleta	15
2.7. Descripción textual de RF07: Dejar bicicleta	17
2.8. Descripción textual de RF08: Reservar	18
2.9. Descripción textual de RF09: Cancelar reserva (explícita)	20
2.10. Descripción textual de RF10: Pagar	21
2.11. Descripción textual de RF11: Actualizar	22
A.1. API para la entidad Estación	79
A.2. API para la entidad Usuario	80
A.3. API para la entidad Reserva	82
B.1. Resultados de las pruebas de validación	85

Capítulo 1

Introducción

1.1. Motivación

Los parques públicos de bicicletas están cada vez más extendidos en las ciudades como alternativa a los medios de transporte tradicionales. Asimismo, los dispositivos móviles representan, para gran parte de la población, la herramienta básica para el manejo de numerosas tareas diarias.

Ambos factores suponen la motivación básica para el desarrollo de una aplicación de gestión de dichos parques, de manera que los usuarios registrados puedan conocer y operar con las estaciones disponibles y se logre una experiencia de usuario positiva al mismo tiempo que se logre un modelo de gestión eficiente.

El presente proyecto surge con la idea de suponer una infraestructura tecnológica base para el desarrollo e investigaciones futuras sobre el área que mejoren la gestión del conjunto de estaciones.

1.2. Objetivos

El objetivo primario del presente proyecto ha sido el desarrollo de una aplicación móvil distribuida para la gestión de un parque público de bicicletas. De manera desglosada:

- Desarrollar una aplicación Android comunicada con un servidor y base de datos mediante servicios web que implemente un sistema de gestión de bicicletas para coger, dejar o reservar bicicletas y anclajes a usuarios registrados.
- Realizar un adecuado entendimiento de las tecnologías disponibles para la implementación de la infraestructura mencionada, como podrían ser protocolos y estándares de comunicación, servidores, sistemas gestores de bases de datos, etc.

- Introducir un primer nivel de adaptación dinámica de precios dependiendo de la disponibilidad individual de cada estación de bicicletas.
- Evitar condiciones de carrera ocasionadas como consecuencia de accesos simultáneos a un mismo recurso.
- Seguir las recomendaciones generales de diseño e implementación para asegurar una aplicación fácilmente mantenible.
- Seguir las normas y recomendaciones generales y específicas de las plataformas utilizadas para el diseño de interfaces de usuario, de modo que se garantice una adecuada experiencia de usuario.
- Cubrir el mayor volumen de dispositivos posible en diferentes ámbitos como el tecnológico (versiones Android), lingüístico (diferentes idiomas), etc.
- Desarrollar una adecuada estrategia de pruebas que asegure una aplicación consistente y libre de errores en sus principales funcionalidades.
- Realizar un estudio conveniente, mediante bibliografía especializada, para la adecuada descripción y documentación del sistema.
- Hacer uso de un sistema controlador de versiones (CVS) que permita tener una traza adecuada del progreso y deje abierta la posibilidad de volver a versiones previas en caso de necesidad.

1.3. Metodología

El modelo de proceso seguido ha sido el *modelo incremental*, que combina elementos de los flujos de proceso lineal y paralelo. Como se puede observar en la figura 1.1, el modelo aplica secuencias lineales en forma escalonada a medida que avanza el calendario de actividades. Cada secuencia lineal produce *incrementos* de software susceptibles de entregarse de manera parecida a los incrementos producidos en un flujo de proceso evolutivo.

Cuando se utiliza un modelo incremental, es frecuente que el primer incremento sea el *producto fundamental*. Es decir, se abordan los requerimientos básicos, pero no se proporcionan muchas características suplementarias (algunas conocidas y otras no). Como resultado del uso y/o evaluación de este primer producto, se desarrolla un plan para el incremento que sigue. El plan incluye la modificación del producto fundamental para cumplir mejor las nuevas necesidades, así como la entrega de características adicionales y más funcionalidad. Este proceso se repite después de entregar cada incremento, hasta terminar el producto final.

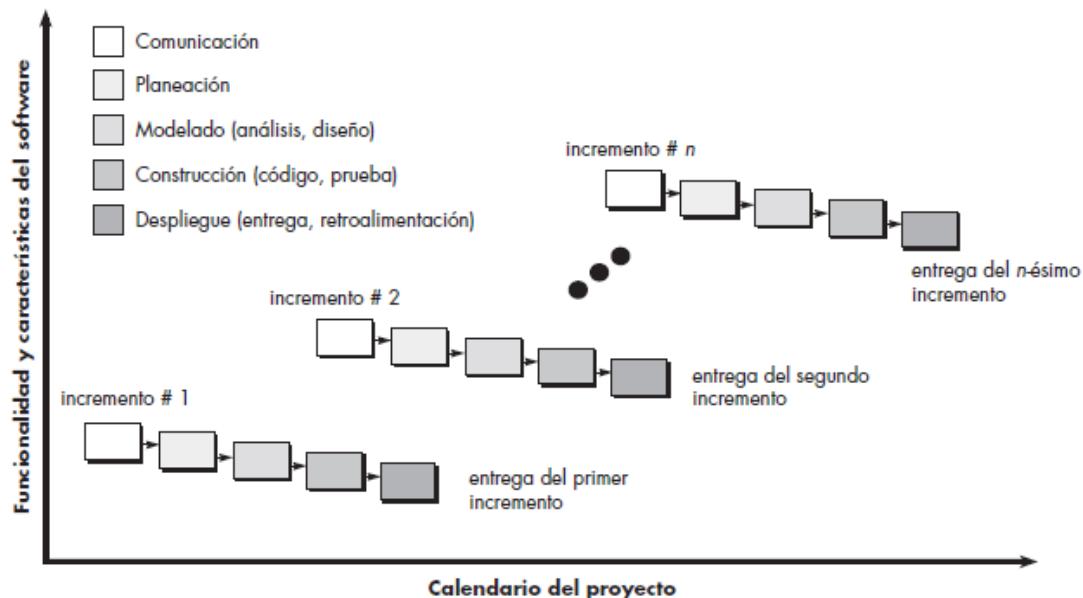


Figura 1.1: El modelo incremental. *Fuente: [Pre10]*

El modelo de proceso incremental se centra en que en cada incremento se entrega un producto que ya opera. Los primeros incrementos son versiones desnudas del producto final, pero proporcionan capacidad que sirve al usuario y también le dan una plataforma de evaluación [Pre10].

Considerando las características anteriores, los incrementos generales establecidos han sido los siguientes:

1. Mapa con estaciones sobre las que poder operar con datos locales.
2. Desarrollo e integración con la aplicación del Servidor y base de datos.
3. Sistema gestor de usuarios.
4. Sistema de gestión de reservas ligadas al usuario.
5. Listado con el detalle de las estaciones mostradas en el mapa.
6. Gráficos con el estado del parque de estaciones.

Capítulo 2

Descripción informática

2.1. Especificación de Requisitos Software

2.1.1. Introducción

Propósito

El propósito de esta sección es el de presentar los requisitos de la aplicación acorde al estándar *IEEE Std. 830-1998: Especificación de Requisitos Software* (ERS en adelante), mostrando y esquematizando la funcionalidad básica del software desarrollado.

El documento va principalmente dirigido a futuros usuarios y desarrolladores de la aplicación, de modo que cuenten con una aproximación teórica a la misma y a sus posibilidades.

Ámbito del Sistema

El sistema, de nombre *Bikesmanager*, consiste en una aplicación Android distribuida encargada de la gestión de parques públicos de bicicletas a través de usuarios previamente registrados.

Se ha buscado desarrollar una aplicación intuitiva y fácil de usar que cubra las necesidades de manera eficiente y eficaz, proporcionando una adecuada experiencia al usuario final.

Definiciones, Acrónimos y Abreviaturas

- ERS: Especificación de Requisitos Software.
- Bikesmanager: Nombre de la aplicación a desarrollar.

- Android: Sistema operativo diseñado principalmente para dispositivos móviles con pantalla táctil.
- Usuario: Persona que hará uso de la aplicación.
- RF: Requerimiento Funcional.
- RNF: Requerimiento No Funcional.

Referencias

- IEEE Std. 830-1998: Especificaciones de los Requisitos del Software [IE830].

Visión General del Documento

Una vez realizada la introducción general previa, se aportará a continuación una primera descripción general del sistema desarrollado para, finalmente, pasar a detallar los requisitos específicos del mismo.

En el apartado dedicado a la descripción general, se aporta una visión global de la aplicación, así como de las funciones básicas de la misma. Dichas funciones se detallan en la sección siguiente, junto con otros aspectos como los atributos del sistema sobre los que se implanta el desarrollo.

2.1.2. Descripción General

Perspectiva del Producto

La aplicación *Bikesmanager* es un producto diseñado para trabajar sobre dispositivos móviles con sistema operativo Android, donde los datos quedan almacenados en una base de datos a la que se accede mediante el servidor de la aplicación. La conexión de la herramienta con el servidor se realiza mediante servicios web.

Funciones del Producto

En la figura 2.1 se aporta el diagrama de casos de uso que muestra, a grandes rasgos, las funciones del sistema.

Características de los Usuarios

- Tipo de usuario: Usuario
 - Nivel educacional: irrelevante.
 - Experiencia técnica: experiencia en el manejo de *smartphones*.
 - Actividad: manejo de la aplicación.

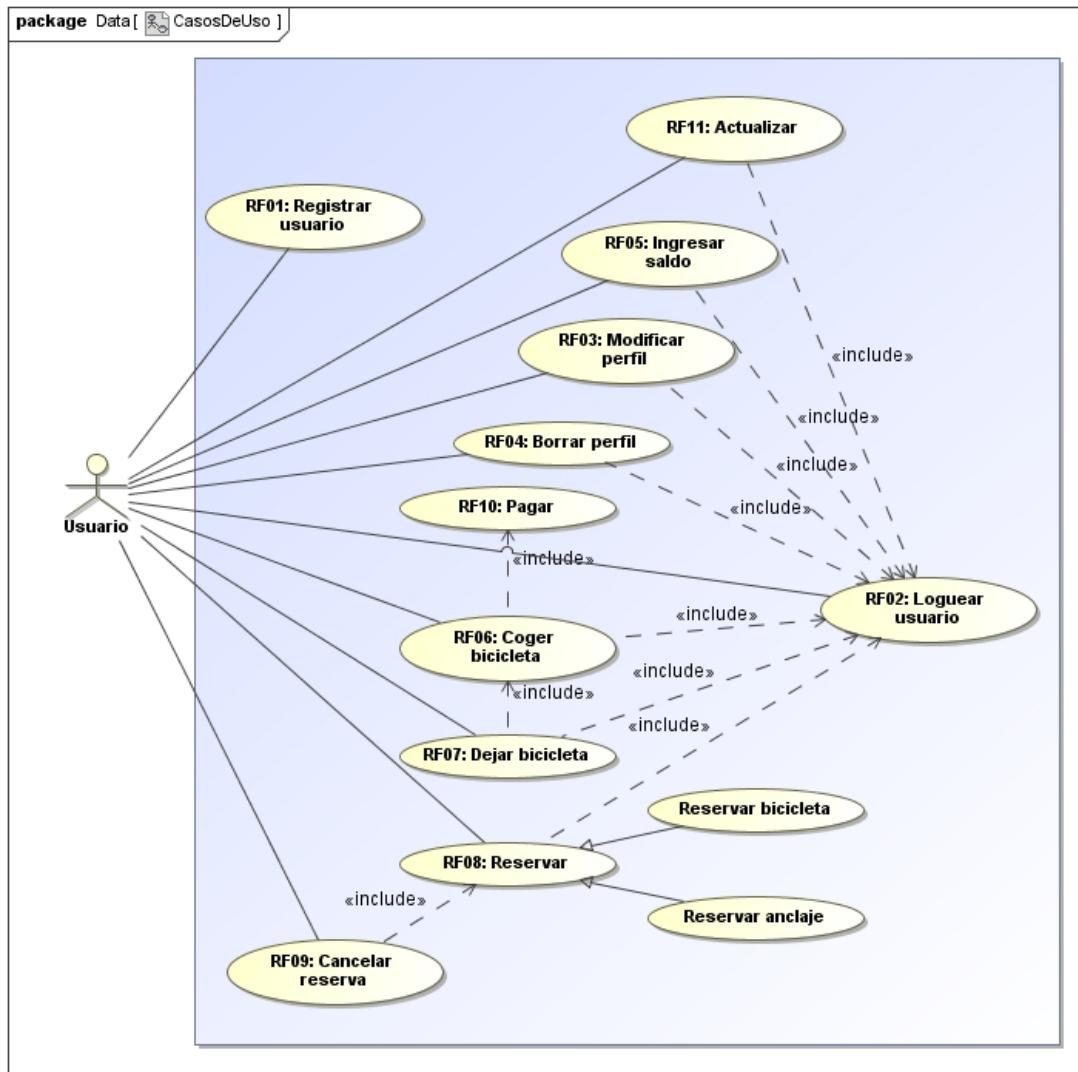


Figura 2.1: Diagrama de casos de uso

Restricciones

- Limitaciones hardware:
 - Los servidores han de ser capaces de atender consultas concurrentes.
 - Los dispositivos móviles han de estar gestionados por el sistema operativo Android.
- Arquitectura del sistema: Cliente/Servidor de tres capas con base datos MySQL

- Lenguaje(s) en uso: JAVA, SQL.
- Protocolos de comunicación:
 - Aplicación – Servidor: tecnología Web Services.
 - Servidor – Base de Datos: MySQL Connector/JDBC.
- Consideraciones acerca de la seguridad:
 - El acceso a la aplicación se realiza mediante el par usuario-contraseña.
 - Las claves de usuario se almacenan de manera segura mediante encriptación SHA-1.
 - Desde la aplicación, ningún usuario tiene acceso a la administración interna de la misma, sino que dicha tarea se realiza directamente sobre el servidor o base de datos.

Suposiciones y Dependencias

- Se asume que los requisitos aquí descritos son estables.
- Los equipos en los que se vaya a ejecutar el sistema deben cumplir los requisitos antes indicados para garantizar el adecuado funcionamiento de la aplicación.

Requisitos Futuros

No se consideran requisitos futuros.

2.1.3. Requisitos Específicos

Interfaces Externas

La interfaz con el usuario consiste en un conjunto de pantallas con los controles habituales: botones, campos de texto, listas, etc. sobre la que se ha tratado de asegurar una adecuada experiencia de usuario mediante diseños que sigan las normas Android¹. Esta interfaz ha sido construida para ser visualizada mediante dispositivos móviles y su diseño se ha fundamentado en los llamados *fragments* de Android, de modo que pueda ser reutilizable y fácilmente transportable a otras dimensiones u orientaciones de pantalla.

¹Los detalles al respecto se pueden consultar en <https://developer.android.com/design/index.html>, a partir de [AnDev].

En relación a las interfaces hardware-software, se ha de contar con un dispositivo móvil con conexión a internet y que implemente, como mínimo, la versión 4.1 del sistema Android (llamada *Jelly Bean*²).

Finalmente, acerca de las interfaces de comunicación, la aplicación se comunica con su servidor mediante el protocolo HTTP haciendo uso de RESTful Web Services y cadenas JSON. La conexión entre el servidor de aplicación y la base de datos se realiza mediante las tecnologías MySQL Connector/JDBC.

Funciones

En esta sección se desarrolla la descripción de los diferentes requisitos funcionales y no funcionales con que deberá contar el sistema.

Se comienza por los *Requerimientos Funcionales*, para los que se aporta su definición acompañada de una descripción textual y del diagrama de actividad correspondiente, de modo que cada uno quede adecuadamente descrito.

- **RF01: Registrar usuario.** El sistema debe permitir el registro de nuevos usuarios. Para ello, se solicitan el nombre usuario, contraseña, dirección de correo electrónico y nombre y apellidos reales. El nombre de usuario y la dirección de correo han de ser únicos, de modo que no se permita la duplicidad de los mismos en la aplicación. Asimismo, la contraseña ha de ser encriptada a la hora de ser almacenada en la base de datos.

La descripción textual:

RF01: Registrar usuario	
Usuario	Sistema
1. Seleccionar registro	
	2. Mostrar interfaz
3. Introducir datos	
	4. Comprobar datos obligatorios
	5. Validar campos únicos
	6. Crear registro

Tabla 2.1: Descripción textual de RF01: Registrar usuario

Y el diagrama de actividad:

²Android denomina alfabéticamente y con nombres de dulces a cada una de las versiones de su sistema operativo. En el momento de la redacción de este documento, la versión más actualizada es la 7, *Nougat*.

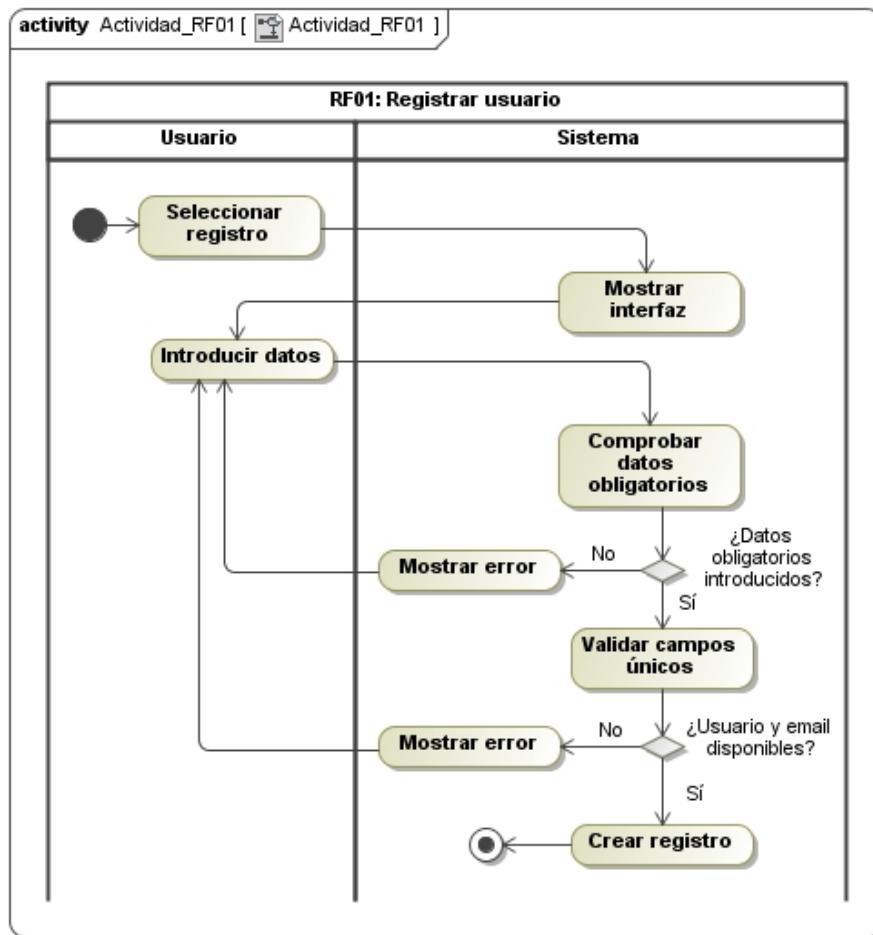


Figura 2.2: Diagrama de actividad de RF01: Registrar usuario

- **RF02: Loguear usuario.** El sistema debe permitir la autenticación de usuarios para su acceso. Este proceso se realiza mediante el par usuario-contraseña. Señalar que, como se ha especificado en el requisito previo, la contraseña se almacena encriptada, de modo que la introducida por el usuario que trata de acceder ha de ser tratada por el mismo algoritmo para poder realizar la comparación.

La descripción textual:

RF02: Loguear usuario	
Usuario	Sistema
1. Seleccionar login	
	2. Mostrar interfaz
3. Introducir datos	
	4. Comprobar campos obligatorios
	5. Autenticar
	6. Completar login

Tabla 2.2: Descripción textual de RF02: Loguear usuario

Y el diagrama de actividad:

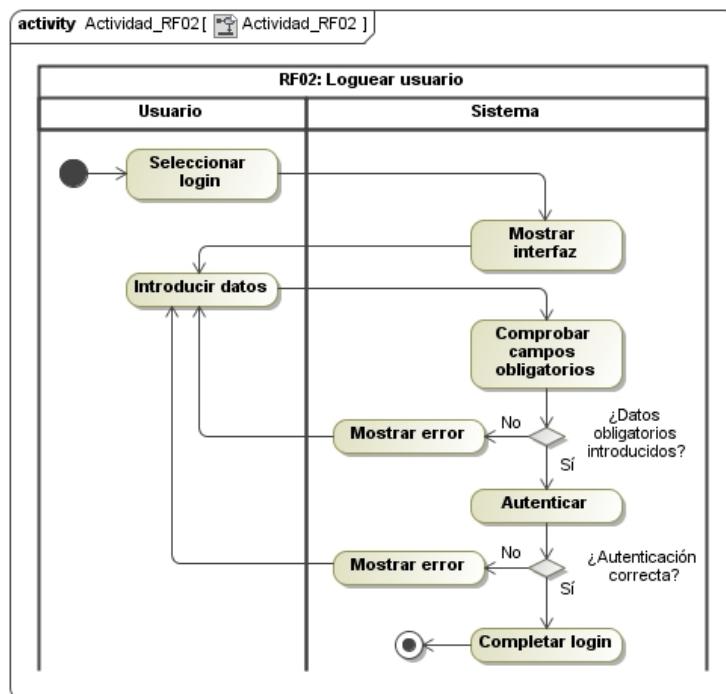


Figura 2.3: Diagrama de actividad de RF02: Loguear usuario

- **RF03: Modificar perfil.** El sistema debe permitir la modificación de los datos del perfil de usuario solicitados en el proceso de registro, considerando que el nombre de usuario y la dirección de correo han de ser únicos.

La descripción textual:

RF03: Modificar perfil	
Usuario	Sistema
[Pto. inclusión: RF02: Loguear usuario]	
1. Seleccionar modificar perfil	2. Mostrar interfaz
3. Introducir datos	4. Validar campos únicos 5. Solicitar confirmación
6. Confirmar operación	7. Actualizar registro 8. Confirmar operación terminada

Tabla 2.3: Descripción textual de RF03: Modificar perfil

Y el diagrama de actividad:

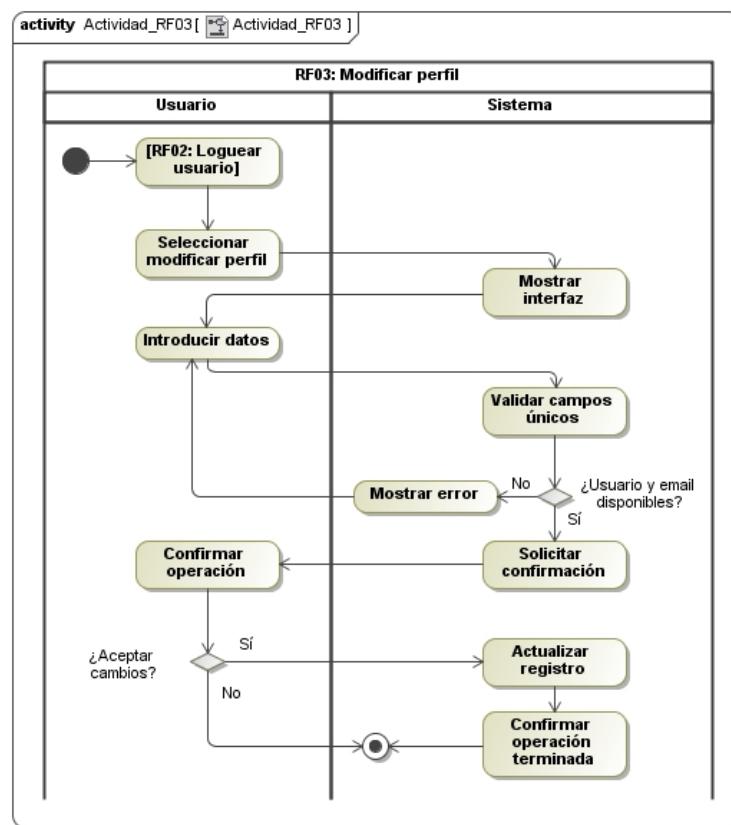


Figura 2.4: Diagrama de actividad de RF03: Modificar perfil

- **RF04: Borrar perfil.** El sistema debe permitir el borrado completo de perfiles. Este proceso requiere de una confirmación explícita por parte del usuario y, en el momento en que se realice, es irreversible, borrando el registro específico de la base de datos. Se devuelve el dinero ingresado y las posibles reservas se cancelan. El proceso no se realiza mientras que el usuario cuente con una bicicleta del conjunto de estaciones en su poder.

La descripción textual:

RF04: Borrar perfil	
Usuario	Sistema
[Pto. inclusión: RF02: Loguear usuario]	
1. Seleccionar borrar perfil	2. Solicitar confirmación
3. Confirmar operación	4. Devolver saldo y eliminar reservas
	5. Borrar registro

Tabla 2.4: Descripción textual de RF04: Borrar perfil

Y el diagrama de actividad:

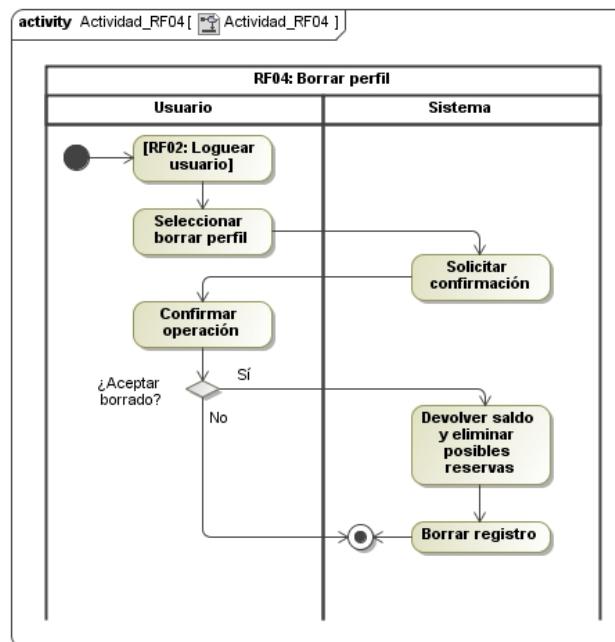


Figura 2.5: Diagrama de actividad de RF04: Borrar perfil

- **RF05: Ingresar saldo.** El sistema debe permitir el ingreso de dinero, de modo que el usuario cuente con saldo suficiente para que pueda operar con el parque de bicicletas.

La descripción textual:

RF05: Ingresar saldo	
Usuario	Sistema
[Pto. inclusión: RF02: Loguear usuario]	
1. Seleccionar ingreso	2. Mostrar interfaz
3. Introducir datos	4. Actualizar registro
	5. Confirmar operación terminada

Tabla 2.5: Descripción textual de RF05: Ingresar saldo

Y el diagrama de actividad:

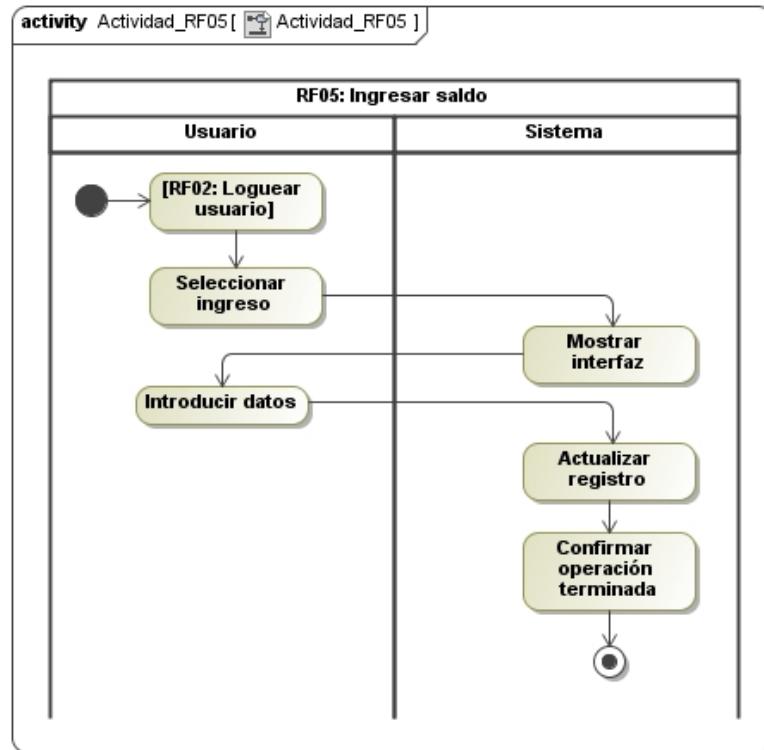


Figura 2.6: Diagrama de actividad de RF05: Ingresar saldo

- **RF06: Coger bicicleta.** El sistema debe permitir coger bicicletas a los usuarios registrados y logueados en el sistema. Se han tener en cuenta las siguientes restricciones:

- El usuario sólo puede tener una bicicleta cogida a la vez.
- El usuario ha de disponer de saldo suficiente para completar la operación.
- En caso de haber reservado previamente, el usuario sólo podrá coger la bicicleta de la estación en la que haya realizado dicha reserva.
- Las bicicletas reservadas por otros usuarios no están disponibles.

Señalar, a su vez, que la estación muestra una tarifa adaptada a la disponibilidad de bicicletas. Es decir, partiendo de una tarifa base, ésta se ve acrecentada a medida que las bicis disponibles se reduzcan. El objetivo es introducir un primer nivel de adaptación a las condiciones del entorno, de modo que los usuarios tengan alicientes para coger bicicletas de estaciones con una mayor disponibilidad.

La descripción textual:

RF06: Coger bicicleta	
Usuario	Sistema
[Pto. inclusión: RF02: Loguear usuario]	
1. Seleccionar estación	
	2. [RF11: Actualizar]
3. Seleccionar coger bici	
	4. Comprobar restricciones
	5. [RF10: Pagar]
	6. Actualizar estado global
	7. Confirmar operación terminada

Tabla 2.6: Descripción textual de RF06: Coger bicicleta

Y el diagrama de actividad:

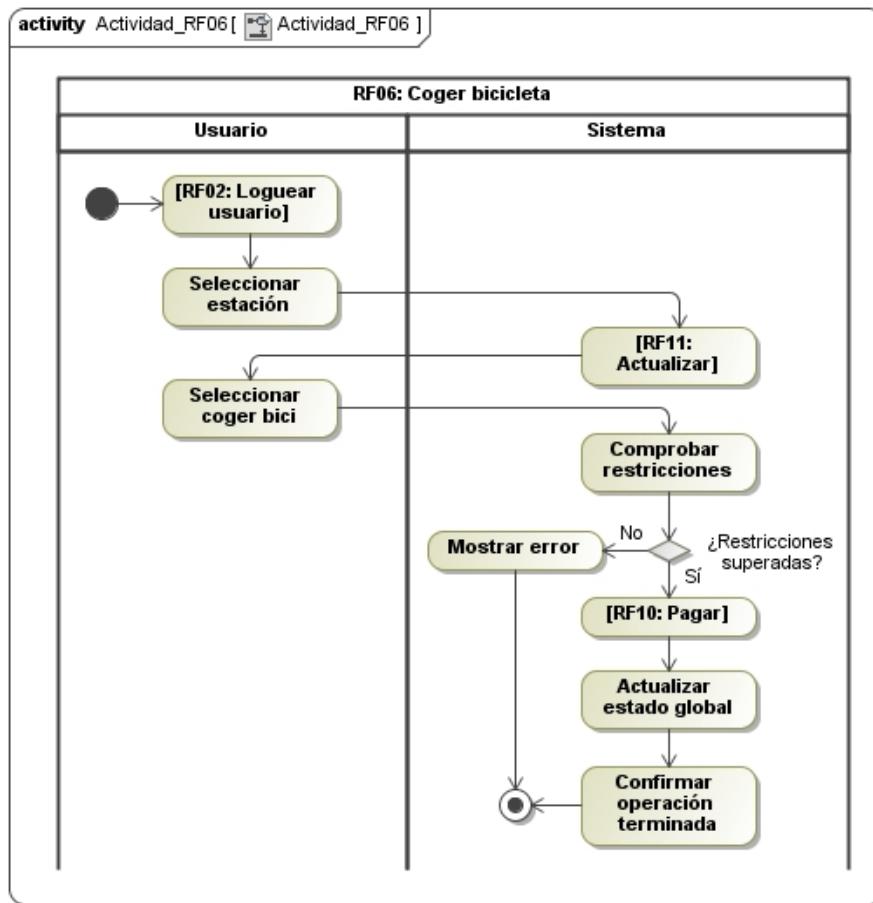


Figura 2.7: Diagrama de actividad de RF06: Coger bicicleta

- **RF07: Dejar bicicleta.** El sistema debe permitir dejar, en estaciones con disponibilidad suficiente, bicicletas previamente cogidas por usuarios. Se han tener en cuenta las siguientes restricciones:

- En caso de haber reservado un anclaje previamente, el usuario sólo puede dejar la bicicleta en la estación en la que haya realizado dicha reserva.
- Los anclajes reservados por otros usuarios no están disponibles.

La descripción textual:

RF07: Dejar bicicleta	
Usuario	Sistema
[Pto. inclusión: RF02: Loguear usuario]	
1. Seleccionar estación	
	2. [RF11: Actualizar]
3. Seleccionar dejar bici	
	4. Comprobar restricciones
	5. Actualizar estado global
	6. Confirmar operación terminada

Tabla 2.7: Descripción textual de RF07: Dejar bicicleta

Y el diagrama de actividad:

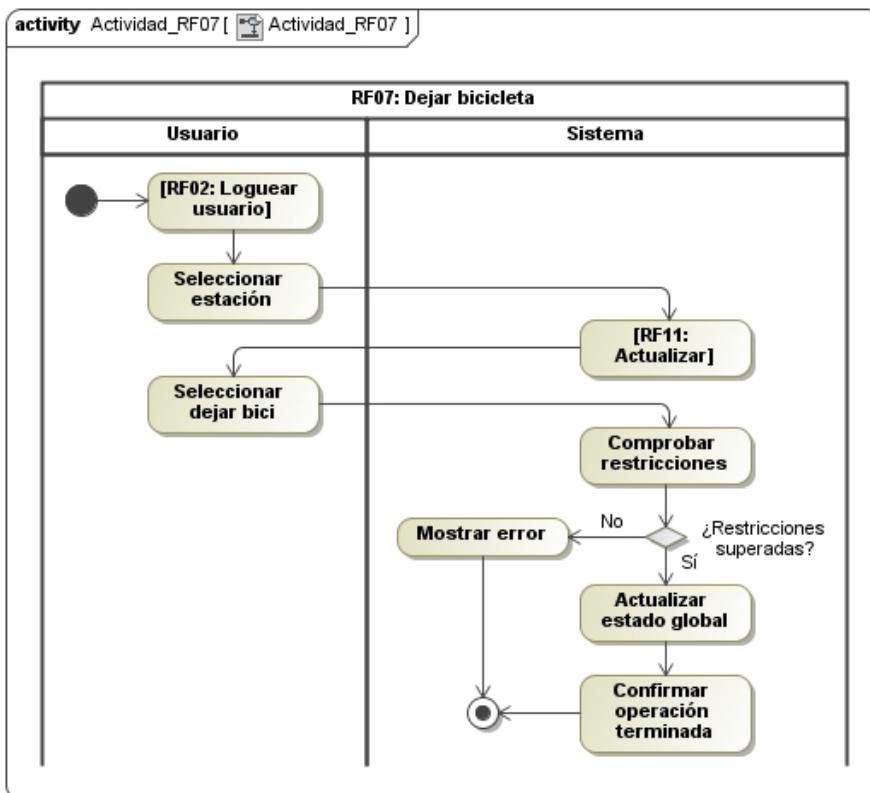


Figura 2.8: Diagrama de actividad de RF07: Dejar bicicleta

- **RF08: Reservar.** El sistema debe permitir la reserva de recursos, que pueden ser de dos tipos: bicicletas o anclajes. Se han tener en cuenta las siguientes restricciones:

tes restricciones:

- El usuario sólo puede tener una reserva de cada tipo a la vez. Es decir, se permite tener, como máximo, dos reservas simultáneas por usuario: una bicicleta y un anclaje.
- En caso de tener una bicicleta, el usuario no puede realizar la reserva de otra.

La descripción textual:

RF08: Reservar	
Usuario	Sistema
[Pto. inclusión: RF02: Loguear usuario]	
1. Seleccionar estación	
	2. [RF11: Actualizar]
3. Seleccionar reservar	
	4. Mostrar opciones de reserva
5. Seleccionar tipo de reserva	
	6. Comprobar restricciones
	7. Actualizar estado global
	8. Confirmar operación terminada

Tabla 2.8: Descripción textual de RF08: Reservar

Y el diagrama de actividad:

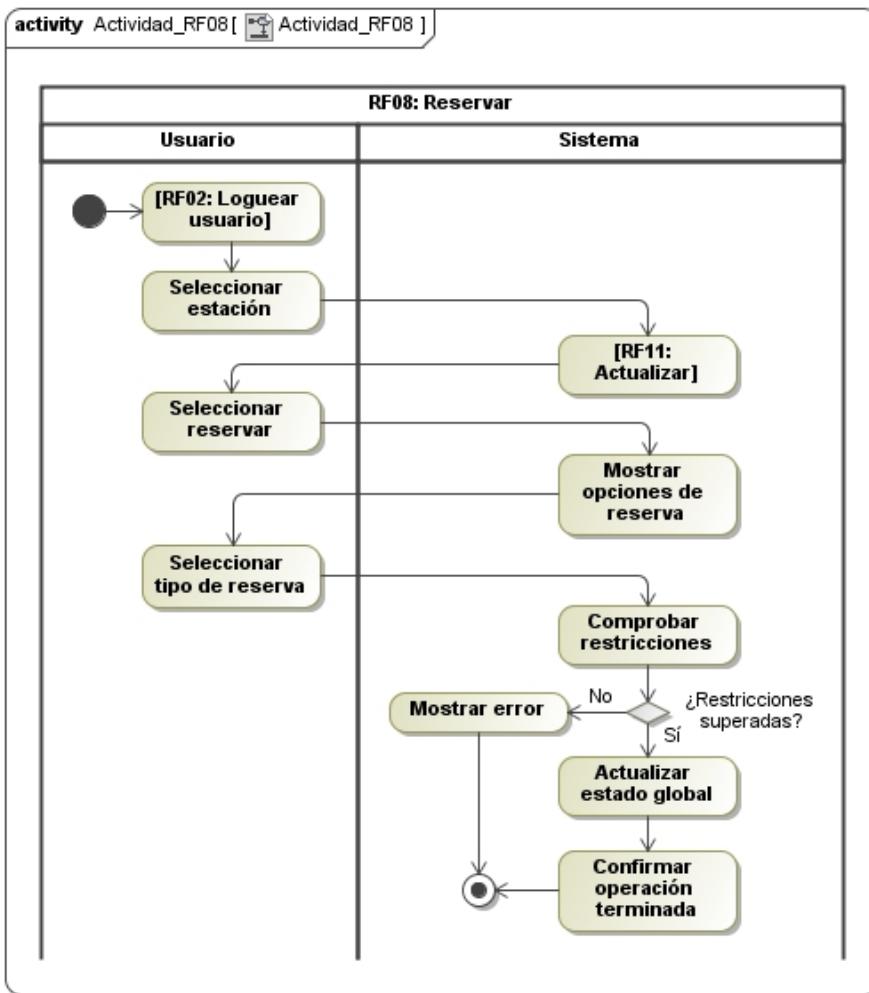


Figura 2.9: Diagrama de actividad de RF08: Reservar

- **RF09: Cancelar reserva.** El sistema debe permitir la cancelación de reservas, tanto de bicicletas como de anclajes. Esta cancelación se puede realizar por las siguientes vías:
 - Cancelación explícita por parte del usuario. Este tipo requiere una confirmación previa.
 - Cancelación implícita al cabo de 30 minutos del momento de la reserva.

Si un usuario ejerce su derecho sobre una reserva, la misma queda automáticamente cancelada. Del mismo modo, si un usuario borra su perfil, sus posibles reservas también se liberan, como se ha comentado antes,

Se aporta la descripción textual del requisito referido a la cancelación explícita, puesto que es la única en la que interviene el usuario de manera directa:

RF09: Cancelar reserva (explícita)	
Usuario	Sistema
[Pto. inclusión: RF08: Reservar]	
1. Seleccionar cancelar reserva	2. Solicitar confirmación
3. Confirmar operación	4. Actualizar estado global
	5. Confirmar operación terminada

Tabla 2.9: Descripción textual de RF09: Cancelar reserva (explícita)

Y el diagrama de actividad:

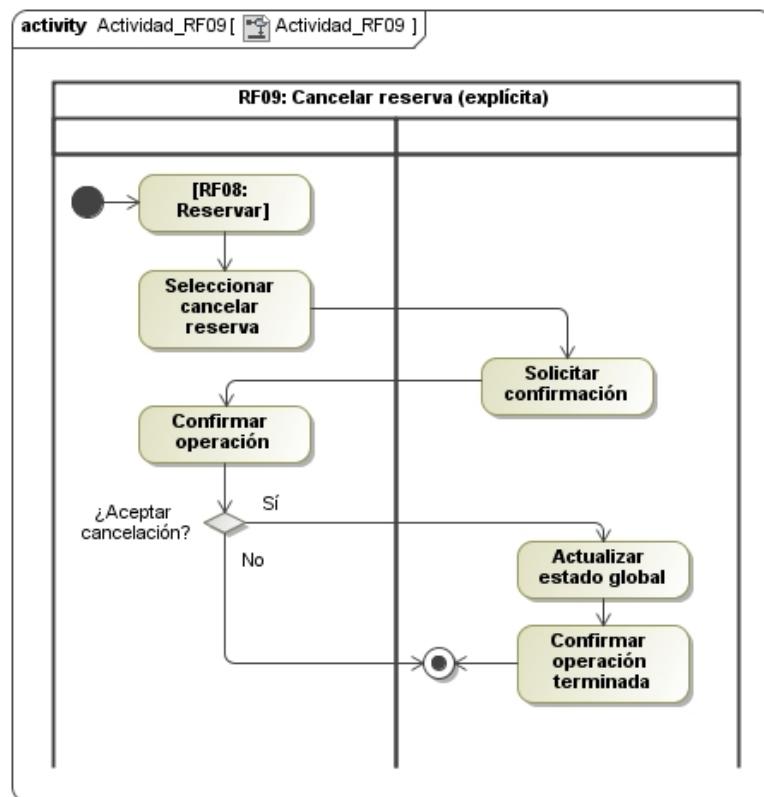


Figura 2.10: Diagrama de actividad de RF09: Cancelar reserva (explícita)

- **RF10: Pagar.** El sistema debe permitir realizar el pago de las operaciones desarrolladas por los usuarios.

La descripción textual:

RF10:Pagar	
Usuario	Sistema
[Pto. inclusión: RF06: Coger bicicleta]	
	1. Realizar pago
	2. Actualizar usuario
	3. Confirmar operación terminada

Tabla 2.10: Descripción textual de RF10: Pagar

Y el diagrama de actividad:

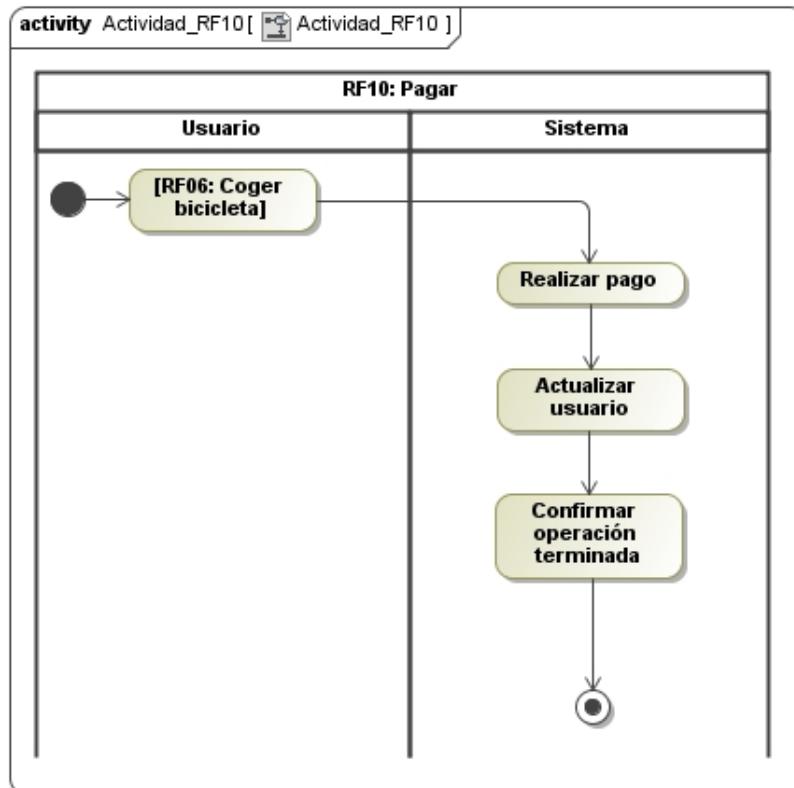


Figura 2.11: Diagrama de actividad de RF10: Pagar

- **RF11: Actualizar.** El sistema debe permitir actualizar el estado global tomando la información más reciente almacenada en el servidor. En este sentido, con cada solicitud de actualización, en el proceso se debe comprobar si existen reservas caducadas, cancelándolas automáticamente para ofrecer al usuario una visión consistente de la disponibilidad del parque de bicicletas.

La descripción textual:

RF11:Actualizar	
Usuario	Sistema
[Pto. inclusión: RF02: Loguear usuario]	
1. Seleccionar actualizar	
	2. Actualizar estado
	3. Mostrar estado actualizado

Tabla 2.11: Descripción textual de RF11: Actualizar

Y el diagrama de actividad:

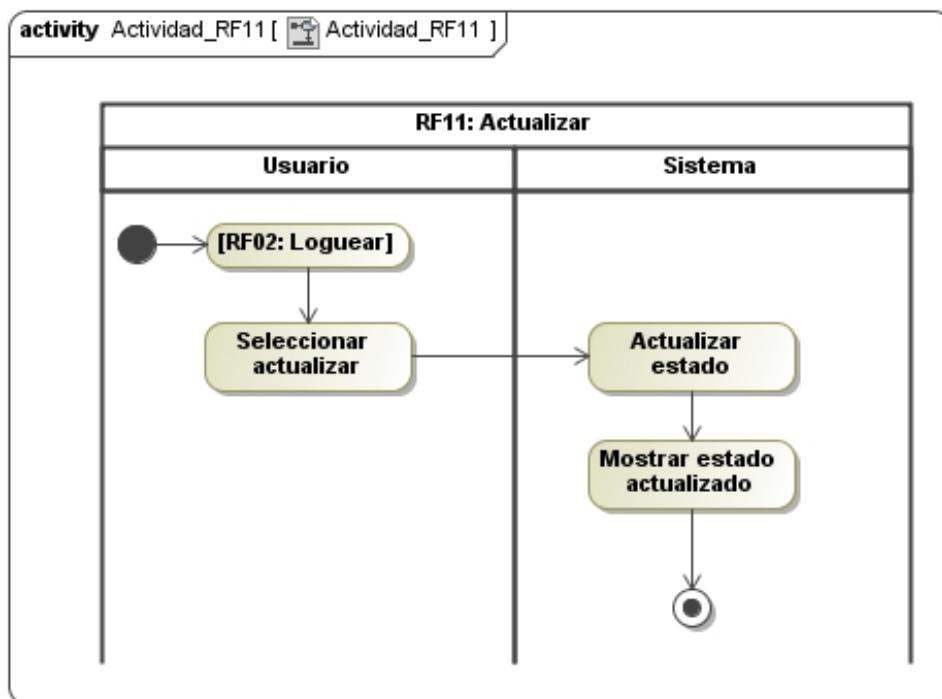


Figura 2.12: Diagrama de actividad de RF11: Actualizar

Finalmente, se pasan a detallar los *Requerimientos No Funcionales* más relevantes:

- **RNF01: Usabilidad.** El sistema ha de presentar una interfaz sencilla e intuitiva, respetando las normas de diseño Android y proporcionando una adecuada retroalimentación al usuario de las operaciones terminadas y de posibles errores en su manejo.
- **RNF02: Eficiencia.** El sistema ha de trasladar toda operación posible a la capa de servidor, de modo que el cliente suponga la menor carga posible para el dispositivo móvil y se reduzcan los tiempos de espera.
- **RNF03: Seguridad.** El sistema ha de asegurar la seguridad en el tratamiento de la información del usuario.

El detalle de este RNF se recoge en el apartado siguiente, referido a los Atributos del Sistema.

- **RNF04: Fiabilidad.** El sistema ha operar según lo esperado y minimizando en la medida de lo posible la probabilidad de aparición de fallos.

El detalle de este RNF se recoge en el apartado siguiente, referido a los Atributos del Sistema.

- **RNF05: Mantenibilidad.** El sistema se ha de desarrollar de modo que pueda ser conservado y restituido (en caso necesario) con facilidad.

El detalle de este RNF se recoge en el apartado siguiente, referido a los Atributos del Sistema.

- **RNF06: Portabilidad.** El sistema ha de desarrollarse de modo que pueda ser ejecutado en diferentes plataformas.

El detalle de este RNF se recoge en el apartado siguiente, referido a los Atributos del Sistema.

- **RNF07: Disponibilidad.** El sistema ha de permanecer en funcionamiento continuo para prestar una servicio adecuado a los usuarios.

El detalle de este RNF se recoge en el apartado siguiente, referido a los Atributos del Sistema.

Este conjunto de RNF suponen una adaptación tomada de los factores de calidad más extendidos como son los Factores de Calidad de McCall [McC77], de Garvin [Gar87] o de la norma ISO 25010 (antigua 9126) [ISO25].

Requisitos de Rendimiento

Desde el punto de vista de la aplicación móvil, no se espera una carga excesiva en el dispositivo por gestionar únicamente los datos del usuario registrado.

El servidor, por su parte, ha de ser capaz de dar respuesta a una serie de peticiones concurrentes que pueden escalar a la cantidad de dispositivos que tengan instalada la aplicación.

Finalmente, en relación a la cantidad de registros almacenados en la base de datos, se espera que queden registrados de manera individual tanto usuarios como puestos de bicicletas, además de posibles tablas adicionales.

Restricciones de Diseño

Para el diseño de la aplicación se deben utilizar componentes compatibles con la versión 4.1 de Android. En caso de no estar disponibles, se ha de recurrir a librerías de soporte para dar servicio a dicha versión. Cumpliendo con ella, las restricciones hardware quedan, a su vez, cubiertas.

En relación al servidor de aplicación, éste ha de ser capaz de gestionar accesos concurrentes a recursos compartidos, controlando las posibles condiciones de carrera que puedan aparecer y dando una respuesta oportuna al usuario.

Atributos del Sistema

- Seguridad.
 - El acceso a la aplicación se realiza mediante el par usuario-contraseña.
 - Las claves de usuario deben almacenarse de manera segura mediante encriptación SHA-1.
 - Desde la aplicación ningún usuario tendrá acceso a la administración interna de la misma, sino que dicha tarea se realizará directamente sobre el servidor o base de datos donde sólo el desarrollador o mantenedor de la aplicación podrá acceder.
- Fiabilidad.
 - El sistema ha de quedar adecuadamente testeado previa distribución para obtener una adecuada cobertura de fallos y errores.
 - La interfaz de usuario ha de ser sencilla e intuitiva y, en caso de ser necesario, debe informar con el resultado de las operaciones para una adecuada experiencia en su uso.
- Mantenibilidad.

- La aplicación ha de quedar desarrollada siguiendo las buenas prácticas del desarrollo software (código adecuadamente organizado y comentado, utilización de patrones de diseño estandarizados, utilización de *idioms*, etc.).
 - La herramienta ha de dejar *logs* internos de las áreas que se consideren más relevantes o susceptibles a fallos, de modo que puedan ser consultados por los desarrolladores para realizar tareas de depuración.
- Portabilidad.
- La aplicación podrá ser instalada en cualquier dispositivo con sistema operativo Android (versión mínima 4.1).
 - Los servidores de aplicación y base de datos se podrán desplegar sobre cualquiera de los sistemas operativos más extendidos (Windows, Linux, iOS, etc.).
 - Queda como tarea futura la adaptación de la aplicación móvil a entornos web e iOS.
- Disponibilidad.
- Los servidores han de estar operativos 24x7 para atender las necesidades de uso.

Otros Requisitos

No se consideran otros requisitos de los especificados en secciones previas.

2.2. Análisis

2.2.1. Introducción

El análisis de los requerimientos descritos ha de dar como resultado la especificación de las características operativas del software, indicando su interfaz y otros elementos del sistema, así como estableciendo las restricciones que limitan al software.

Este proceso de análisis otorga la información que se traduce en diseños de arquitectura, interfaz y componentes, así como los medios para evaluar la calidad del software una vez construido.

Por lo tanto, el modelo de análisis supone un puente entre la descripción del sistema y su diseño posterior. Dicha relación queda esquematizada en la figura 2.13.

Este modelo se desarrollará en dos pasos:

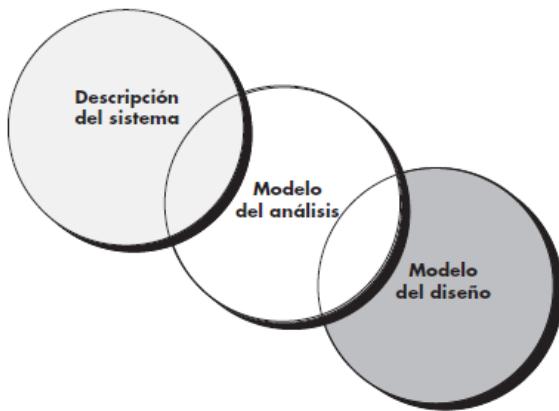


Figura 2.13: El modelo de análisis como puente entre los requerimientos y el diseño del software. *Fuente: [Pre10]*

1. **Análisis conceptual del sistema.** Se presentará la vista estática general del sistema mediante un diagrama de clases de análisis. El objetivo es el de obtener una perspectiva global y con un alto nivel de abstracción del software y sus entidades básicas.
2. **Análisis y desarrollo de requisitos.** Los requisitos descritos en la sección anterior se comenzarán a desarrollar mediante diagramas de colaboración (o comunicación), de modo que se tenga una primera visión de la implementación de cada uno y se muestre una primera aproximación a la interacción entre entidades.

Cabe señalar que, en este punto, la atención se centra en el *qué*, no en el *cómo*, con lo no se entrará en detalles técnicos, de diseño o de implementación.

2.2.2. Análisis conceptual del sistema

Para el análisis y representación del modelo conceptual de datos se recurrirá a diagrama de clases de análisis, que de lugar a la vista estática general del sistema e identifique sus entidades básicas usando la terminología del dominio aplicable. Si bien se cuenta con un cierto grado de subjetividad, las clases potenciales a incluir en este punto deberán cubrir todos (o casi todos) los puntos siguientes [Coa91]:

1. *Información retenida.* Debe recordarse la información sobre la clase para que el sistema pueda funcionar.
2. *Servicios necesarios.* La clase potencial debe tener un conjunto de operaciones identificables que cambien en cierta manera el valor de sus atributos.

3. *Atributos múltiples.* Centrando la atención en la información “principal”.
4. *Atributos comunes.* Para la clase potencial, se define un conjunto de atributos que se aplica a todas las instancias de la clase.
5. *Operaciones comunes.* Se define un conjunto de operaciones para la clase potencial que se aplica a todas sus instancias.
6. *Requerimientos esenciales.* Las entidades externas que aparezcan en el espacio del problema y que produzcan o consuman información esencial para la operación de cualquier solución para el sistema casi siempre se definirán como clases en el modelo de requerimientos.

Con todo, quedan identificadas las siguientes entidades básicas:

- **Gestor Conexión Remota.** Desde el momento en el que la aplicación cuenta con una arquitectura Cliente/Servidor y la comunicación entre ambas entidades se realiza de manera remota, se requiere de una entidad que gestione dicha comunicación y sepa dar respuesta a las operaciones CRUD³.
- **Entidad Remota.** La entidad que queda gestionada por la clase anterior, supone la generalización de alguna de las siguientes entidades:
 - **Usuario.** Se deben recoger los atributos básicos para poder realizar el login, como el nombre de usuario y contraseña, datos personales, como el nombre y apellidos, y datos operativos, como el saldo o datos acerca de si se tiene una reserva o bicicleta. Del mismo modo, el usuario ha de ser capaz de interactuar con las estaciones de bicicletas, cogiendo y dejando las mismas o gestionando reservas.
 - **Estación.** La estación de bicicletas debe contar con datos identificativos, como la dirección donde se encuentre ubicada, y con datos operativos, referidos al total de bicicletas o anclajes disponibles, reservados, etc., así como la tarifa base. La estación ha de poder gestionar las operaciones comenzadas por los usuarios y mostrar su estado convenientemente actualizado.
 - **Reserva.** Se considera relevante contar con una clase para almacenar las reservas de usuarios, de modo que su gestión global sea más sencilla. Esta entidad debe contar con los datos de la reserva (tipo (anclaje o bicicleta), nombre de usuario, fecha y estación) y con la constante que defina el tiempo máximo por reserva (30 minutos, de acuerdo a los requisitos).

³Siglas en inglés de las operaciones básicas para la gestión de datos: Create, Read, Update y Delete

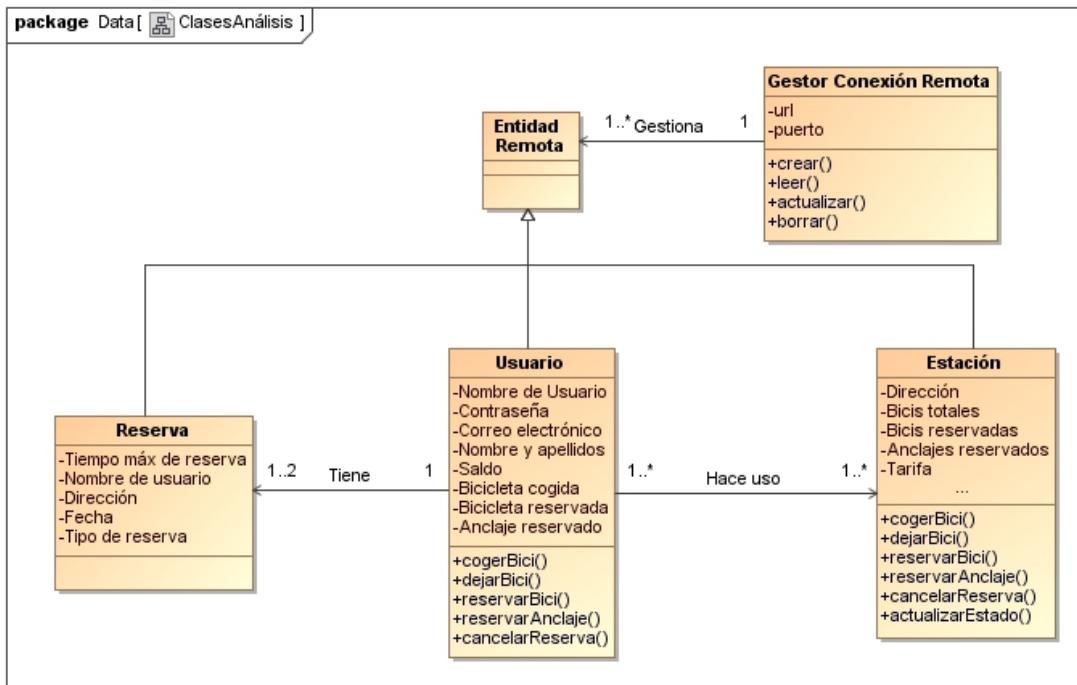


Figura 2.14: Diagrama de clases de análisis

De este modo, el *Gestor de Conexión Remota* es el encargado de enviar y recibir datos para las entidades, sin necesidad de diferenciar entre ninguna de ellas por atender directamente la *Entidad Remota* general .

Este conjunto de entidades básicas, junto con sus potenciales atributos y operaciones básicas, queda modelizado en la figura 2.14.

2.2.3. Análisis y desarrollo de requisitos

Con la vista estática del futuro sistema presentada, a continuación se pasa a describir las realizaciones más relevantes de cada caso de uso presentados en la figura 2.1. Esto es, por cada requisito funcional, se modelizará la realización de su camino principal y caminos alternativos de relevancia obtenidos a partir de los diagramas de actividad anteriores.

El objetivo es el de componer un análisis adecuado de las posibles operaciones e interacciones que cada requisito puede desencadenar y las entidades afectadas ante los caminos de ejecución más probables.

- **RF01: Registrar usuario.** Tres caminos básicos:

1. Camino principal, la operación de registro finaliza satisfactoriamente.
Ver figura 2.15.
2. Camino alternativo 1, el usuario no introduce todos los datos obligatorios. Ver figura 2.16.
3. Camino alternativo 2, se aportan todos los datos requeridos, pero el nombre de usuario o dirección de correo electrónico ya están en uso.
Ver figura 2.17.

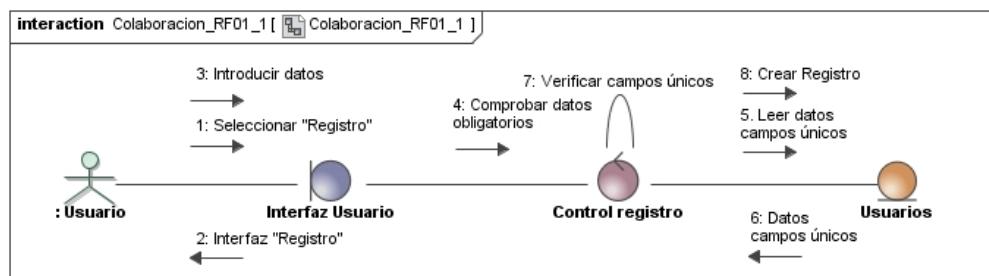


Figura 2.15: Diagrama de colaboración del camino principal del RF01

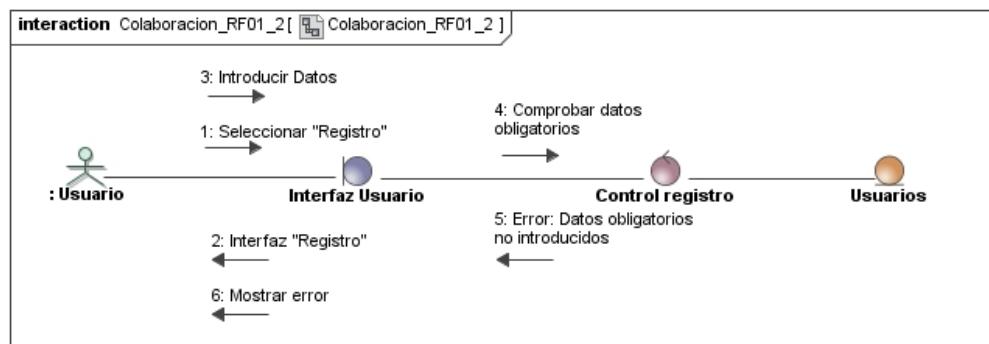


Figura 2.16: Diagrama de colaboración del primer camino alternativo del RF01

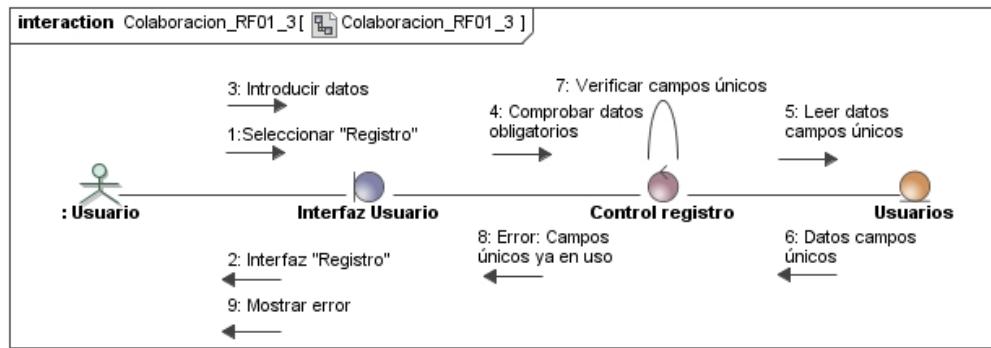


Figura 2.17: Diagrama de colaboración del segundo camino alternativo del RF01

■ **RF02: Loguear usuario.** Tres caminos básicos:

1. Camino principal, la operación de inicio de sesión finaliza satisfactoriamente. Ver figura 2.18.
2. Camino alternativo 1, el usuario no introduce todos los datos obligatorios. Ver figura 2.19.
3. Camino alternativo 2, autenticación incorrecta, se aportan todos los datos requeridos, pero el nombre de usuario y/o contraseña no son correctos. Ver figura 2.20.

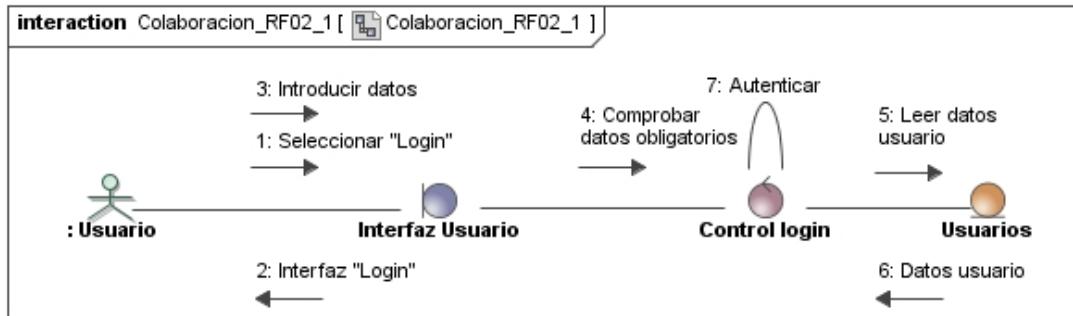


Figura 2.18: Diagrama de colaboración del camino principal del RF02

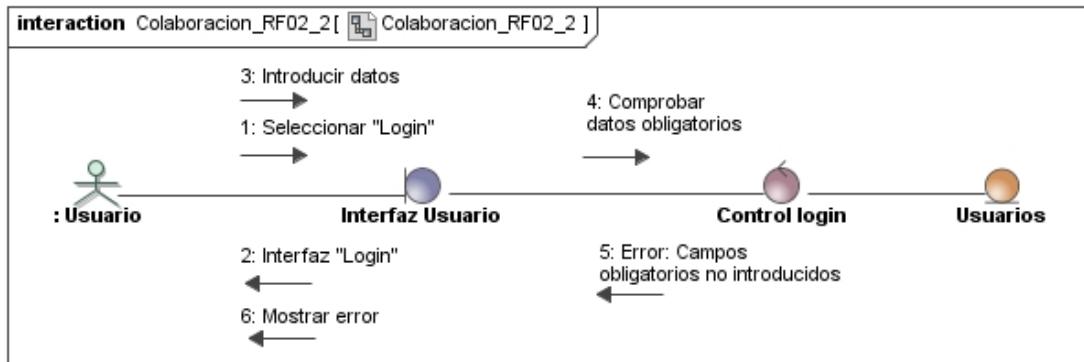


Figura 2.19: Diagrama de colaboración del primer camino alternativo del RF02

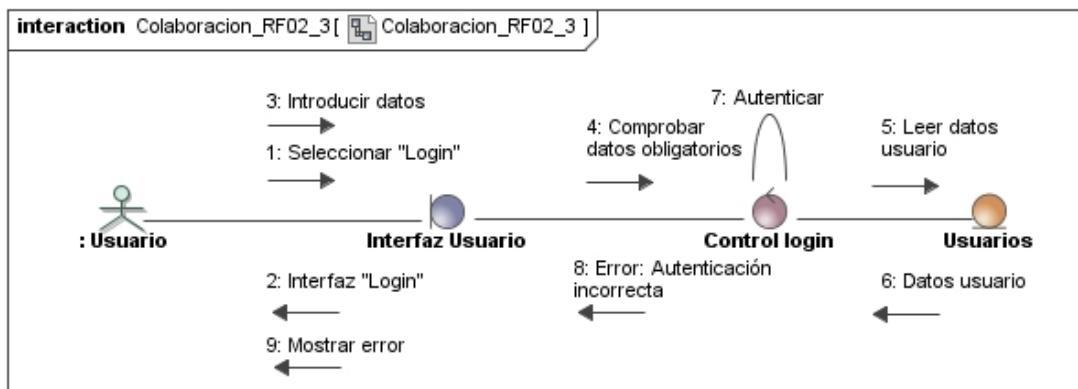


Figura 2.20: Diagrama de colaboración del segundo camino alternativo del RF02

- **RF03: Modificar perfil.** Se consideran tres caminos básicos:

1. Camino principal, la operación de modificación de perfil finaliza satisfactoriamente. Ver figura 2.21.
2. Camino alternativo 1, el nuevo nombre de usuario y/o dirección de correo electrónico ya están en uso. Ver figura 2.22.
3. Camino alternativo 2, el usuario cancela la operación ante la confirmación final. Ver figura 2.23.

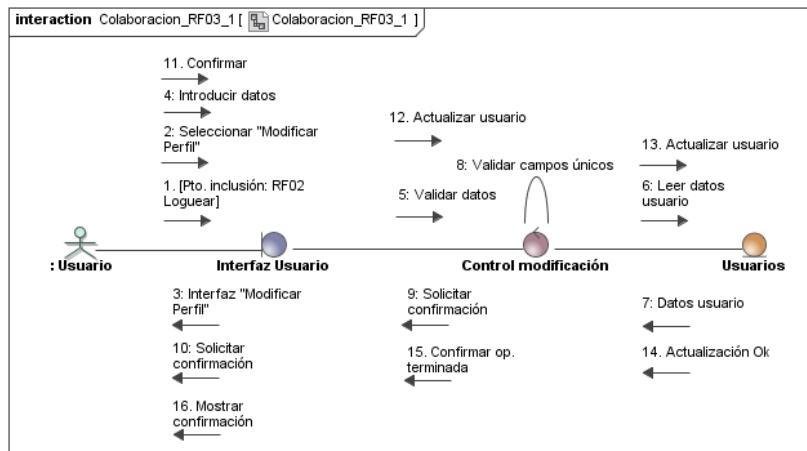


Figura 2.21: Diagrama de colaboración del camino principal del RF03

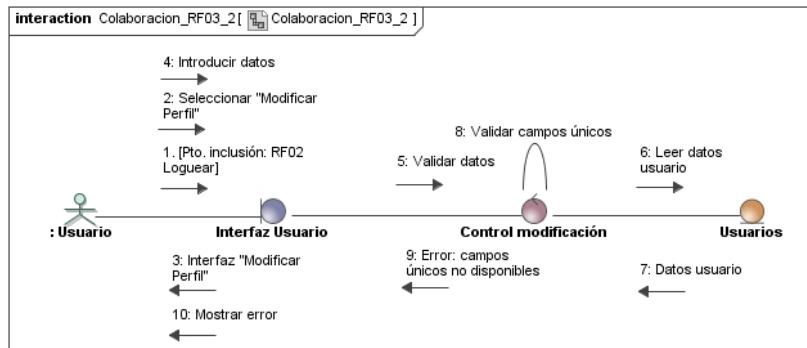


Figura 2.22: Diagrama de colaboración del primer camino alternativo del RF03

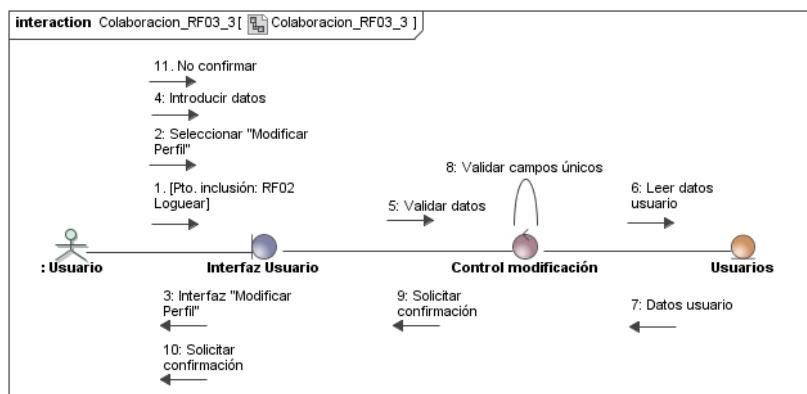


Figura 2.23: Diagrama de colaboración del segundo camino alternativo del RF03

- **RF04: Borrar perfil.** Se consideran dos caminos básicos:

1. Camino principal, la operación de borrado de perfil finaliza satisfactoriamente. Ver figura 2.24.
2. Camino alternativo 1, el usuario cancela la operación ante la confirmación final. Ver figura 2.25.

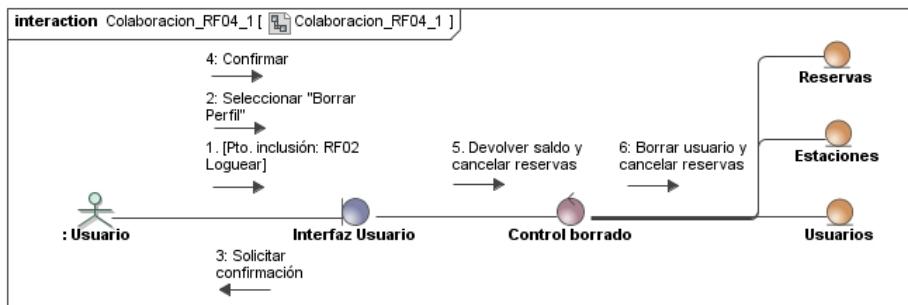


Figura 2.24: Diagrama de colaboración del camino principal del RF04

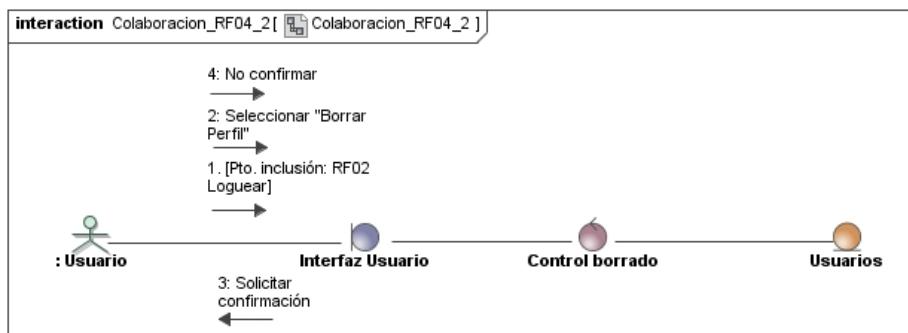


Figura 2.25: Diagrama de colaboración del primer camino alternativo del RF04

- **RF05: Ingresar saldo.** Se considera un único camino básico: el ingreso de saldo sin incidencias. Ver figura 2.26.

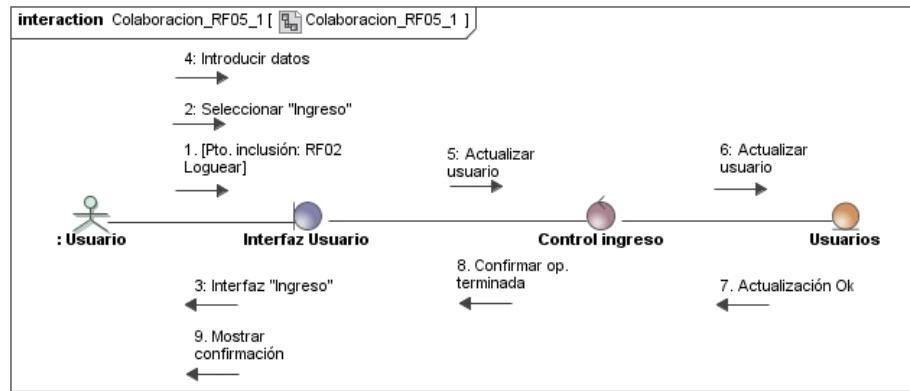


Figura 2.26: Diagrama de colaboración del camino principal del RF05

■ **RF06: Coger bicicleta.** Se consideran dos caminos básicos:

1. Camino principal, la operación finaliza satisfactoriamente. Ver figura 2.27.
2. Camino alternativo 1, el usuario no supera alguna de las restricciones impuestas sobre la operación (mencionadas en la especificación de requisitos anterior). Ver figura 2.28.

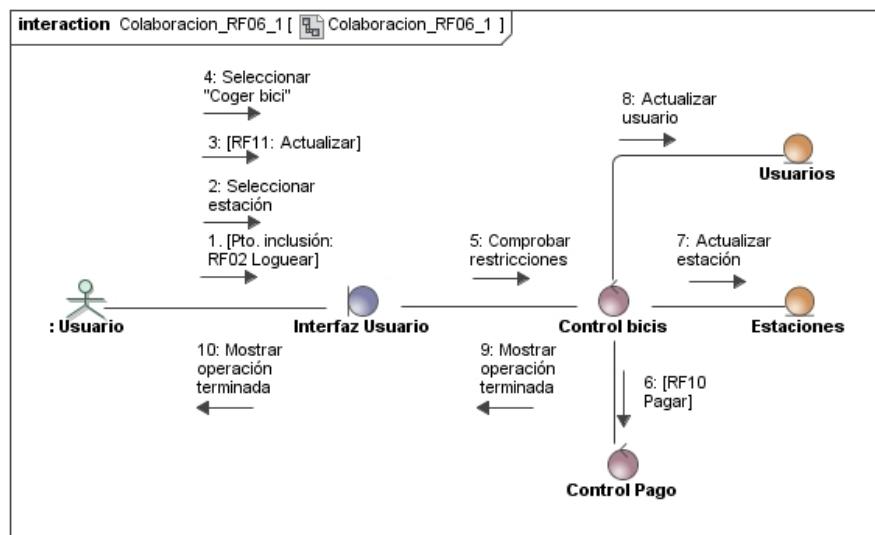


Figura 2.27: Diagrama de colaboración del camino principal del RF06

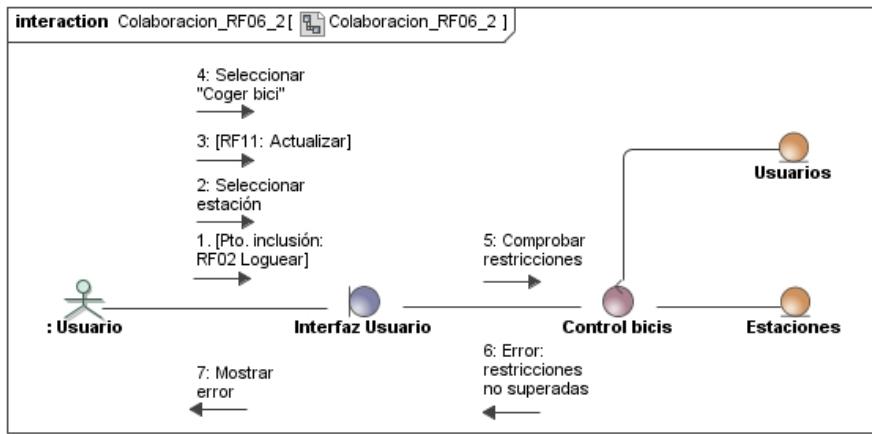


Figura 2.28: Diagrama de colaboración del primer camino alternativo del RF06

■ **RF07: Dejar bicicleta.** Se consideran dos caminos básicos:

1. Camino principal, la operación finaliza satisfactoriamente. Ver figura 2.29.
2. Camino alternativo 1, el usuario no supera alguna de las restricciones impuestas sobre la operación (mencionadas en la especificación de requisitos anterior). Ver figura 2.30.

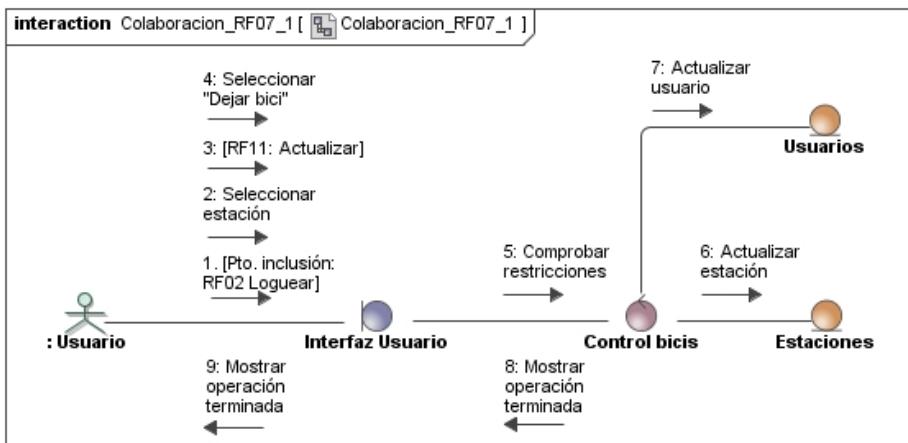


Figura 2.29: Diagrama de colaboración del camino principal del RF07

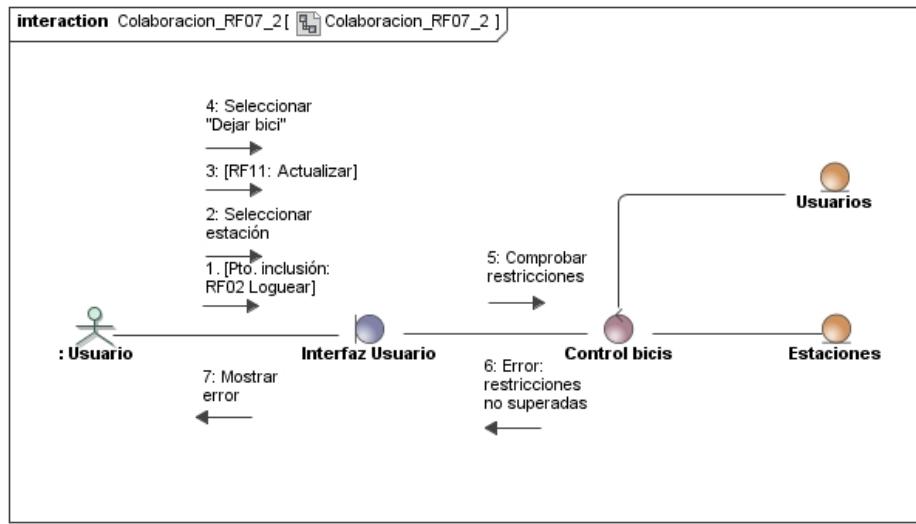


Figura 2.30: Diagrama de colaboración del primer camino alternativo del RF07

■ **RF08: Reservar.** Se consideran dos caminos básicos:

1. Camino principal, la operación finaliza satisfactoriamente. Ver figura 2.31.
2. Camino alternativo 1, el usuario no supera alguna de las restricciones impuestas sobre la operación (mencionadas en la especificación de requisitos anterior). Ver figura 2.32.

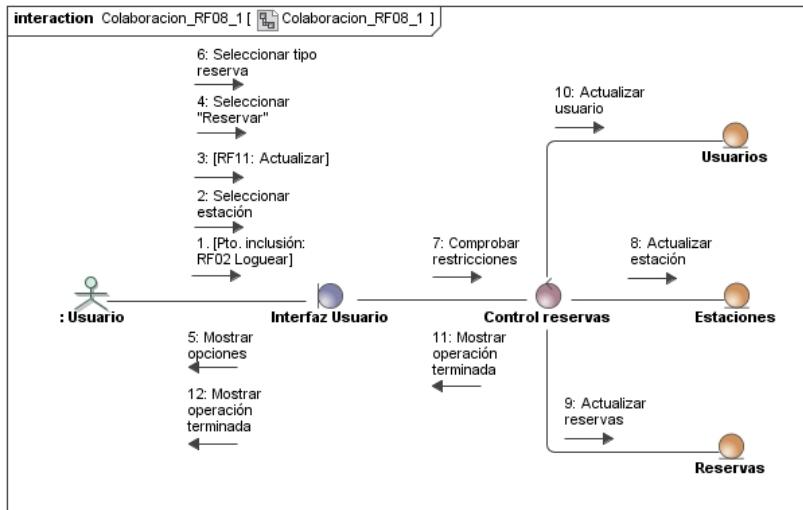


Figura 2.31: Diagrama de colaboración del camino principal del RF08

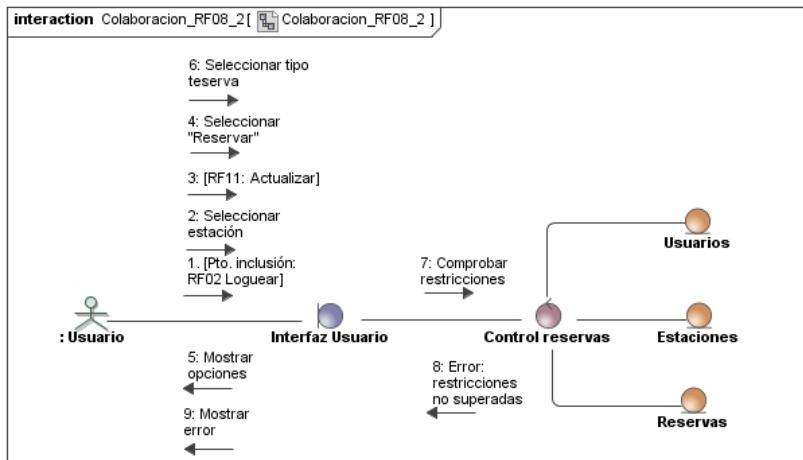


Figura 2.32: Diagrama de colaboración del primer camino alternativo del RF08

- **RF09: Cancelar reserva.** Se consideran dos caminos básicos:

1. Camino principal, la operación finaliza satisfactoriamente. Ver figura 2.33.
2. Camino alternativo 1, el usuario cancelar la operación ante la confirmación final. Ver figura 2.34.

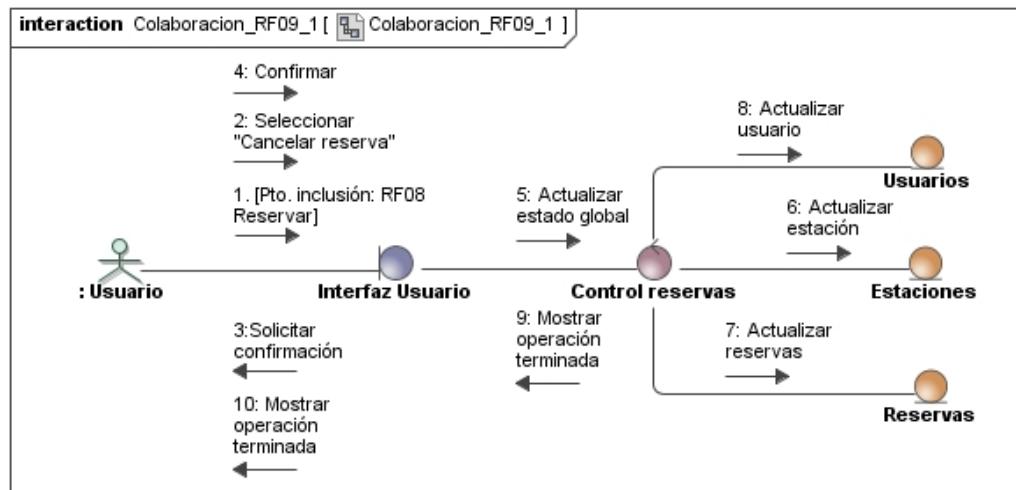


Figura 2.33: Diagrama de colaboración del camino principal del RF09

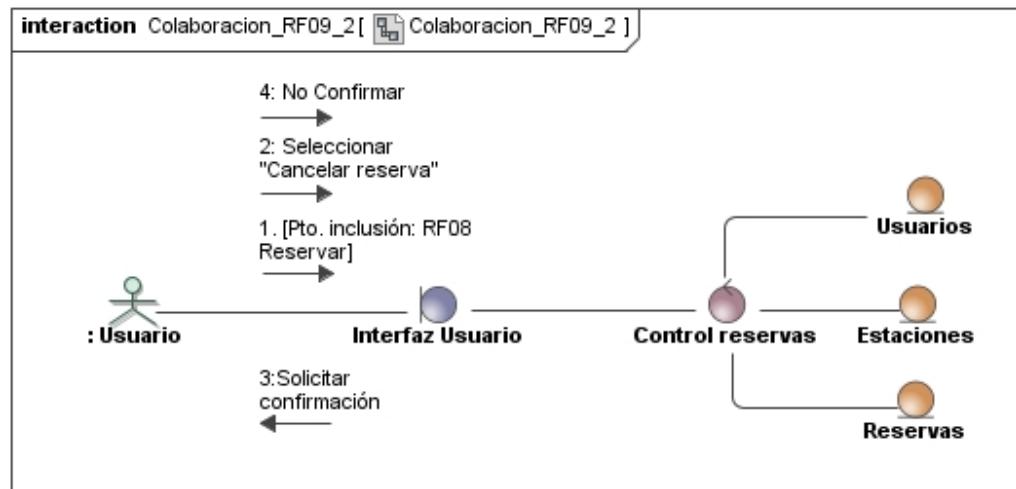


Figura 2.34: Diagrama de colaboración del primer camino alternativo del RF09

■ **RF10: Pagar.** Se considera un camino básico:

1. Camino principal, la operación finaliza satisfactoriamente. Ver figura 2.35.

Dado el carácter académico de la aplicación, el *pago* sólo implica la actualización del saldo disponible para el usuario dentro del sistema, no se considera que dicha operación pueda tener caminos alternativos de relevancia.

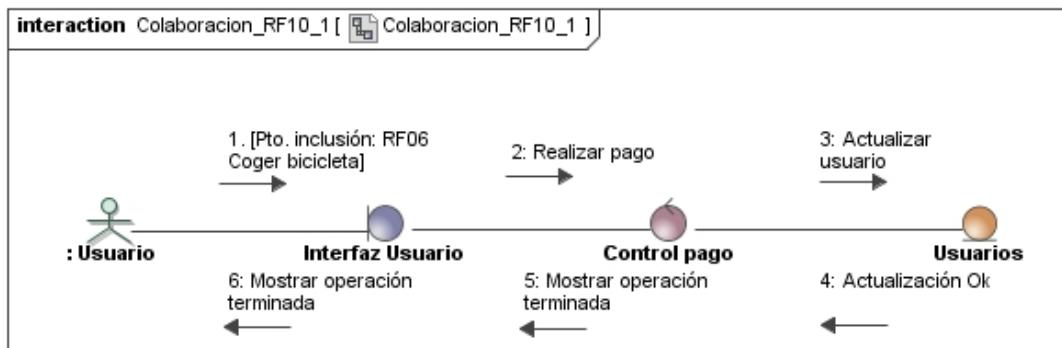


Figura 2.35: Diagrama de colaboración del camino principal del RF10

■ **RF11: Actualizar.** Se consideran dos caminos básicos:

1. Camino principal, en el proceso de actualización se identifican reservas caducadas, que se cancelan automáticamente. Ver figura 2.36.
2. Camino alternativo 1, no se identifican reservas caducadas, el proceso consiste en leer datos actualizados. Ver figura 2.37.

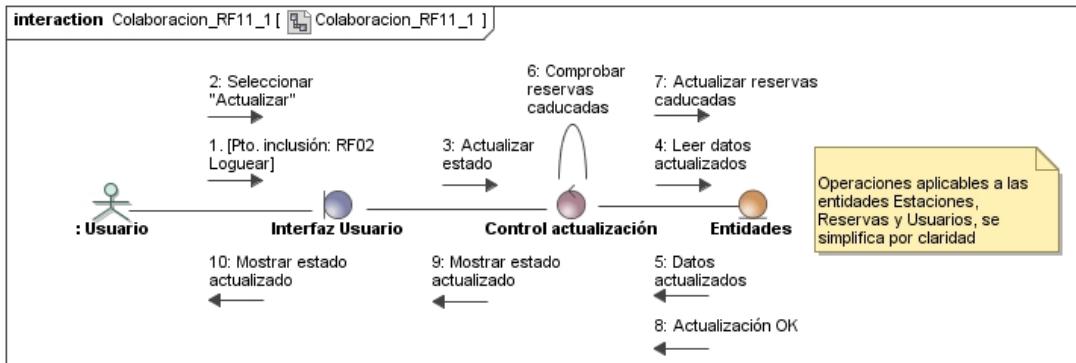


Figura 2.36: Diagrama de colaboración del camino principal del RF11

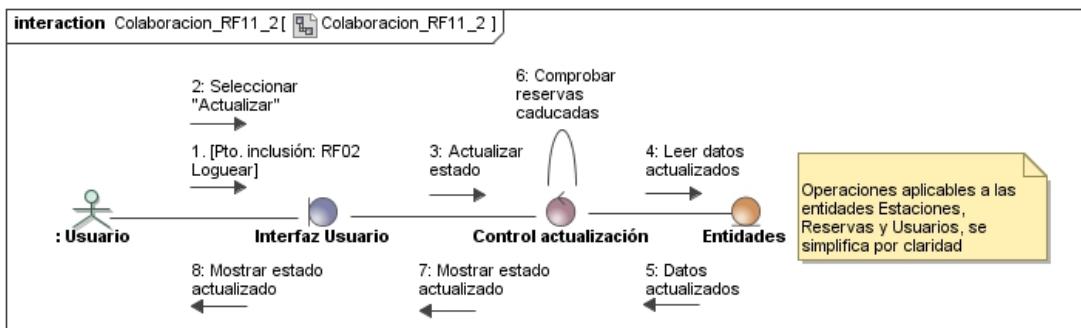


Figura 2.37: Diagrama de colaboración del primer camino alternativo del RF11

2.3. Diseño

2.3.1. Introducción

Sintetizando a Pressman [Pre10], el diseño del software comienza una vez que se han analizado y modelado los requerimientos, etapas centradas en describir los datos que se necesitan, la función y el comportamiento, es la última acción de la ingeniería de software dentro de la actividad de modelado y prepara la etapa de construcción (generación y prueba de código) proporcionando detalles sobre arquitectura del software, estructuras de datos, interfaces y componentes.

Así, el modelo de diseño que a continuación se desarrolla, se dividirá dos partes principales:

- Diseño de la arquitectura.** Se presentará el diseño de la arquitectura general que adoptará el sistema y de las interrelaciones entre sus componentes. La modelización de dicha estructura se llevará a cabo mediante el diagrama de clases de diseño, dando lugar a visión más detallada del sistema a implementar.
- Diseño de las interfaces.** En esta segunda parte se recogerán las decisiones de diseño más relevantes con referencia a las interfaces de usuario y comunicación de la aplicación.

Por su parte, el diseño en el nivel de componentes y despliegue se tratará en la sección 2.4 siguiente, acerca del *Modelo de Implementación*.

2.3.2. Diseño de la arquitectura

La *arquitectura del software* supone “la estructura o estructuras del sistema, lo que comprende a los componentes del software, sus propiedades externas visibles y las relaciones entre ellos” [Bas03].

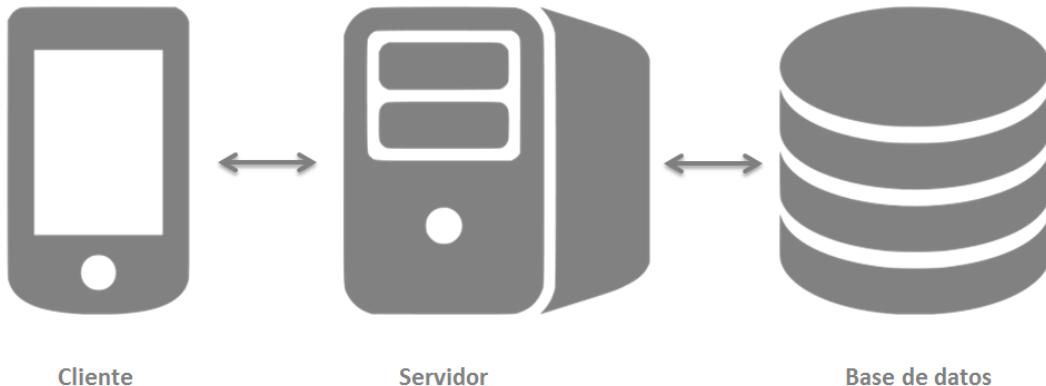


Figura 2.38: Esquema de la arquitectura Cliente/Servidor de tres capas

La arquitectura no es el software operativo, sino una representación que permite [Pre10]: 1) analizar la efectividad del diseño para cumplir los requerimientos establecidos, 2) considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y 3) reducir los riesgos asociados con la construcción del software.

La definición previa pone el énfasis en el papel de los “componentes del software”, que puede ser algo tan simple como un módulo de programa o una clase orientada a objeto, si bien también puede ampliarse a conceptos más extensos y complejos.

Estilo arquitectónico

Considerando los conceptos anteriores y con base en las especificación de requisitos previa, el estilo arquitectónico sobre el que se sustenta el sistema es el *Cliente/Servidor* en su variación de tres capas, tal y como se muestra en la figura 2.38:

1. **Cliente.** Supone la capa de presentación, la Interfaz de Usuario. Su función es traducir tareas y/o resultados para resultar comprensibles por el usuario.
2. **Servidor.** La capa lógica encargada de coordinar la aplicación y procesar datos entre el resto de capas.
3. **Base de datos.** Implica la capa de datos, donde la información es almacenada y recuperada por parte de la capa lógica para su procesamiento.

En la sección 2.4 se entra en mayor detalle sobre estas capas, especificando las tecnologías para la implementación de cada una y la comunicación entre ellas.

Modelización de la arquitectura

En el modelo de análisis desarrollado en la sección 2.2 quedaron definidas un conjunto de clases de análisis (representadas en la figura 2.14). Con un nivel de abstracción relativamente alto, cada una describe algún elemento del dominio del problema y se centra en aspectos de éste visibles para el usuario.

En este punto, se reduce el nivel de abstracción y las clases de análisis se afinan hacia las clases de diseño, aportando los detalles que permitirán que las clases se implementen y generen una infraestructura para el software a desarrollar.

A la hora de realizar este refinamiento, se han de tener en cuenta una serie de características que aseguren que las clases de diseño quedan bien formadas [Arl02]:

1. *Completa y suficiente.* Una clase de diseño debe encapsular todos los atributos y métodos que sea razonable esperar y que existan para la clase.
2. *Primitivismo.* Los métodos asociados con una clase de diseño deben centrarse en el cumplimiento de un servicio para la clase. Una vez implementado el servicio con un método, la clase no debe proveer otro modo de hacer lo mismo.
3. *Mucha cohesión.* Una clase de diseño cohesiva tiene un conjunto pequeño y centrado de responsabilidades; para implementarlas emplea atributos y métodos de objetivo único.
4. *Poco acoplamiento.* Dentro del modelo de diseño, es necesario que las clases de diseño colaboren una con otra. Sin embargo, la colaboración debe mantenerse en un mínimo aceptable. Si un modelo de diseño está muy acoplado (todas las clases de diseño colaboran con todas las demás), el sistema es difícil de implementar, probar y mantener con el paso del tiempo. En general, las clases de diseño dentro de un subsistema deben tener sólo un conocimiento limitado de otras clases.

Se ha de tener en cuenta que el sistema cuenta tanto con una capa de aplicación como con otra de servidor, con lo que se ha de modelizar la arquitectura de ambas capas.

Así, en la figura 2.39 se aporta el diagrama de clases de diseño para la capa del cliente del sistema, si bien cabe señalar que el objetivo del diagrama de clases de diseño es el de modelizar la arquitectura básica del sistema a implementar; con lo que, para mantener la vista de dicha arquitectura útil y clara, no se incluyen clases, atributos o métodos que tengan un carácter accesorio o no aporten de manera directa a la consecución de los requisitos descritos en secciones anteriores.

De este modo, no se incluyen métodos ligados a inicializaciones de interfaces de usuario o atributos relativos a las mismas, métodos básicos como los *get()* y *set()* o clases accesorias sin implicación directa en los casos de uso, entre otros.

Sobre el diagrama presentado, destaca el empleo de dos patrones de diseño:

1. **Singleton.** Patrón creacional utilizado con el objetivo de asegurar que una clase se instancia una única vez.

Dada la naturaleza de la entidad *Usuario* según la cual una ejecución en un dispositivo únicamente puede contar con un usuario activo a la vez (con posibles modificaciones sobre su estado), se considera que este patrón ayuda a forzar dicho objetivo, eliminando posibles errores o fallos futuros.

2. **Observer.** Patrón de comportamiento utilizado para implementar la actualización de un objeto (o varios) ante el cambio en el estado de otro.

Dado que los datos se almacenan por unas entidades y se muestran por otras, es necesario que cada vez que se produzca una interacción con el servidor y éste notifique un resultado, las clases del cliente que provocaron dicha llamada se actualicen convenientemente mediante la modificación de su estado. Con ello, dicho patrón se aplica sobre las clases que provocan llamadas directas al servidor y las que gestionan las mismas. La base del patrón es una interfaz a ser implementada por las clases a ser actualizadas y sobre la que se notificarán los cambios o respuestas del servidor por parte de las clases gestoras de las comunicaciones.

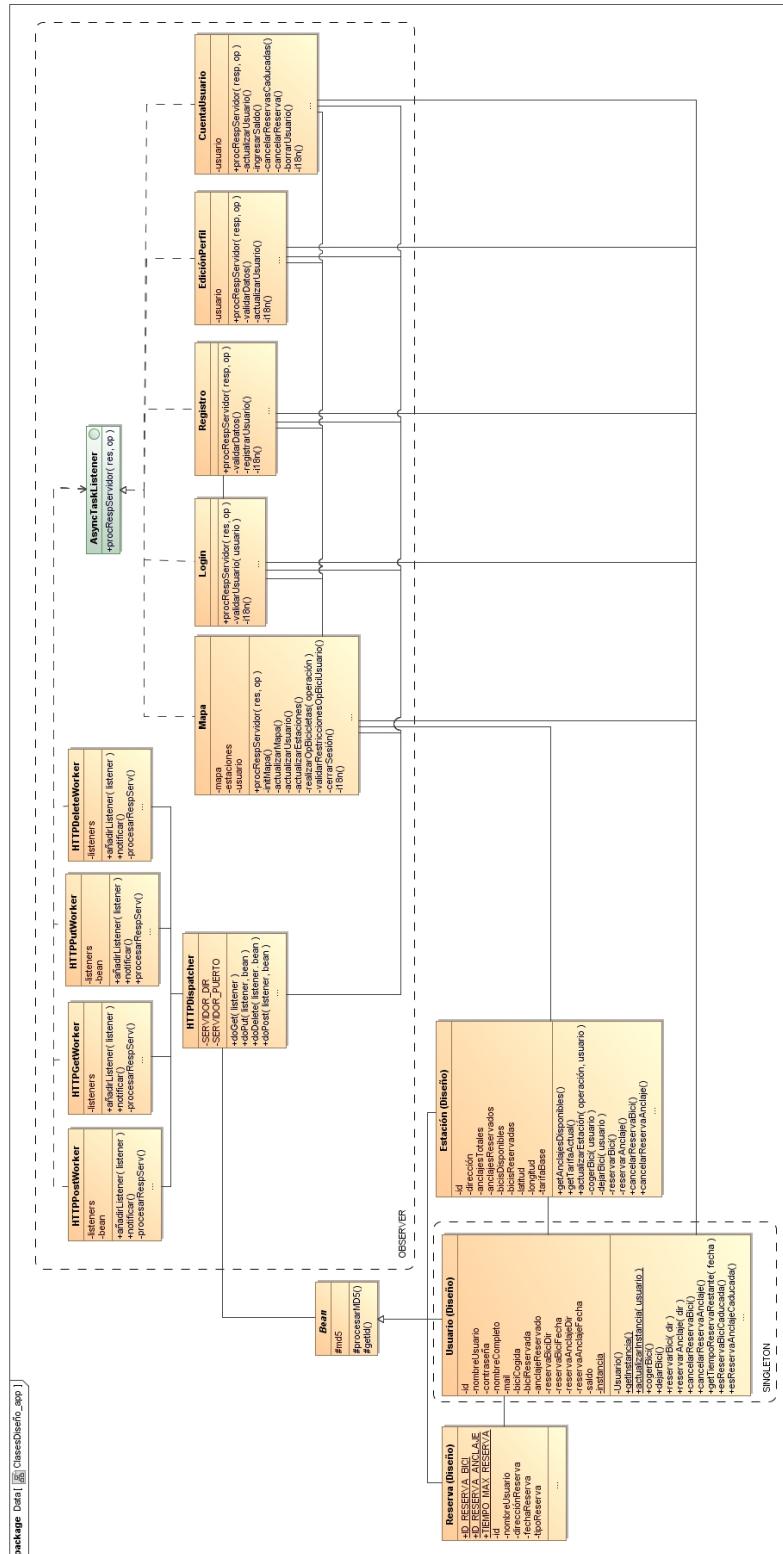


Figura 2.39: Diagrama de clases de diseño de la capa del cliente

Por su parte, en la figura 2.40 queda esquematizada la arquitectura del servidor.

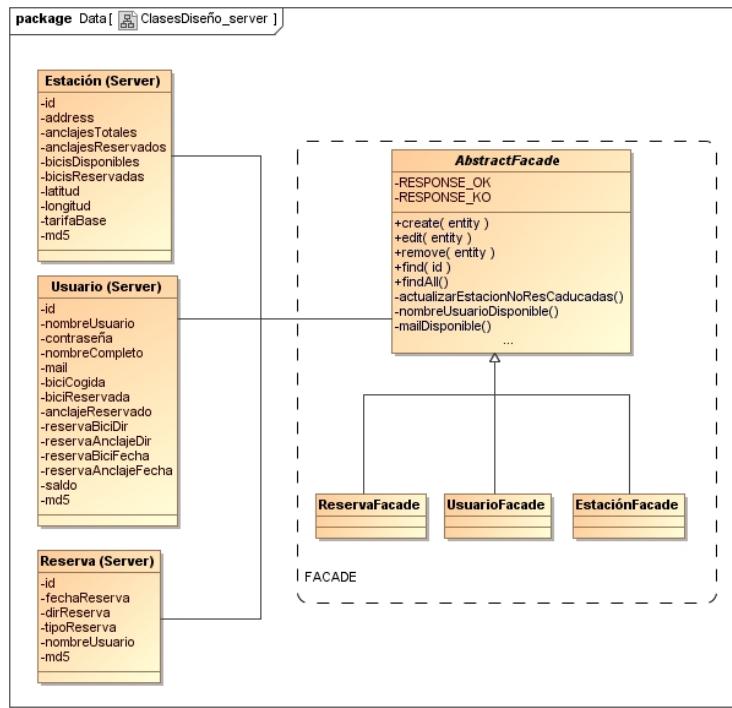


Figura 2.40: Diagrama de clases de diseño de la capa del servidor

En este caso, se identifica el patrón *Facade*, un patrón estructural utilizado para simplificar las interfaces o comunicaciones entre subsistemas.

Dado que el servidor supone la capa intermedia entre el cliente y la base de datos, este patrón ayuda a facilitar la comunicación entre ellos. El patrón se implementa partiendo de un clase abstracta de la que heredan las *fachadas* específicas de cada entidad, siendo estas especificaciones las que reciben las llamadas remotas. Todas ellas cuentan con los métodos CRUD básicos implementados en la fachada padre, que es la que finalmente realiza la conexión con la base de datos. Esta clase padre es la que implementa, a su vez, los métodos de control para todas las entidades, de modo que las fachadas hijas sólo se dedican a recibir llamadas e invocar el método de la clase padre oportuno.

Por claridad en el esquema, en las fachadas hijas no se han incluido los métodos de comunicación básicos por tratarse de llamadas al método CRUD específico de la clase padre. Asimismo, en esta fachada padre se apuntan dichos métodos CRUD sin entrar en posibles adecuaciones a cada entidad, además de métodos operativos necesarios para la funcionalidad del sistema.

Se observa, a su vez, que la fachada padre ha de incluir dos mensajes estándar de respuesta para indicar si la operación ha ido bien o no, de modo que el cliente

(que también conoce estos mensajes) reaccione de manera oportuna.

2.3.3. Diseño de las interfaces

Una vez establecida la arquitectura básica sobre la que se fundamenta el sistema, el segundo apartado a definir en el modelo de diseño son las interfaces de comunicación y de usuario. Estos elementos permiten que el software se comunique externamente y permita la comunicación y colaboración internas entre los componentes que constituyen la arquitectura del software.

Interfaces de comunicación

Dentro de la arquitectura Cliente/Servidor se hace necesario definir la comunicación entre ambas capas. Si bien los detalles técnicos para su implementación se comentan en la siguiente sección 2.4, en este punto se aporta una capa de diseño que suponga una introducción conceptual.

Como quedó definido en la sección 2.1 dedicada a la ERS, la comunicación Cliente/Servidor se ha de regir de acuerdo a la tecnología RESTful Web Services, que se comentará más adelante. Esta tecnología parte de comunicaciones HTTP y alcanzan los diferentes métodos remotos mediante URLs diferenciadas. Así, se hace necesario definir una URL estándar que sirva de base para todas las concretas de cada método a ejecutar en el servidor. Se considera la siguiente plantilla:

http : // [IP-SERV] : [PORT] / BikesManager / rest / entities . [ENTITY]

donde *IP-SERV* indica la IP dirección del servidor, *PORT* indica el puerto donde el servidor se encuentra escuchando y *ENTITY* indica la entidad (Usuario, Estación o Reserva) que realiza la llamada remota.

A partir de esta URL general se forman las específicas de la forma siguiente:

[BASE_URL] / [METHOD]

donde *BASE_URL* es la URL básica indicada y *METHOD* indica el método o parámetros que se añaden en la dirección para la llamada del método que sea necesario.

Una vez un método se ejecuta en el servidor, éste devuelve la cadena resultante al cliente. En caso de que se tenga que indicar si la operación se ha realizado correctamente o no, se envían cadenas del estilo *[ENTITY]_SERVER_OK* o *[ENTITY]_SERVER_KO*, respectivamente.

Por su parte, las cadenas que se intercambian entre ambas capas siguen el formato de texto JSON⁴, la tecnología ha utilizar para la traducción de cadenas se comenta en la siguiente sección 2.4.

⁴Una introducción a este lenguaje se recoge en [JSON].

La selección de este formato en lugar del tradicional XML queda principalmente motivada por el hecho de que el primero es más sencillo a la hora compartir *datos tradicionales* como texto o números, que son los utilizados por el sistema desarrollado, y no datos estructurados como podrían ser documentos, fotos, vídeos... donde XML es más eficiente.

Finalmente, comentar la utilización de los patrones *Observer* y *Facade* para mejorar y estandarizar la comunicación entre componentes ante las problemáticas específicas apuntadas anteriormente.

Finalmente, en el anexo A, se recoge una descripción de la API implementada en el servidor a nivel de las diferentes URLs.

Interfaces de usuario

La interfaz de usuario (IU en adelante) se considera como uno de los elementos más importantes de un sistema o producto basado en computadora. Una interfaz inadecuada perjudica la capacidad del usuario de aprovechar todo el potencial de una aplicación, pudiendo dar lugar al fracaso de un adecuado diseño e implementación.

De acuerdo a Pressman [Pre10], el diseño de la IU (o diseño de la *usabilidad*) crea un medio eficaz de comunicación entre los seres humanos y la computadora. Siguiendo un conjunto de principios de diseño de la interfaz, se identifican los objetos y acciones de ésta y luego crea una plantilla de pantalla que constituye la base del prototipo de la interfaz de usuario.

Este apartado está dedicado, por tanto, a desarrollar brevemente la decisiones y principios de diseño de IU de mayor relevancia adoptados para alcanzar una interfaz gráfica consistente y adecuada en el sistema desarrollado.

En primer término, el diseño general de cualquier IU se ha de basar en las conocidas como *Reglas de Oro* establecidas por Mandel [Man97]:

1. *Dejar el control al usuario.*
2. *Reducir la carga de memoria del usuario.*
3. *Hacer que la interfaz sea consistente.*

Cada una de las tres Reglas anteriores supone una agrupación de reglas más específicas y su estudio teórico queda fuera del alcance de este documento, que se limita a su nombrado para su aplicación práctica⁵.

Con la base anterior establecida, se ha de tener en cuenta que la IU se presenta sobre dispositivos móviles con sistema operativo Android, que aporta numerosa

⁵El detalle de cada regla se puede consultar en el manual de Theo Mandel al respecto en [Man97].

documentación oficial acerca del adecuado diseño de IUs⁶. Se hace necesario, por tanto, hacer un seguimiento de estos principios específicos para lograr una IU adecuada a la naturaleza del dispositivo sobre el que se despliega el sistema.

Adicionalmente, la aplicación debe soportar la ejecución en diferentes idiomas, dependiendo del lenguaje configurado en el dispositivo utilizado. Android ofrece la posibilidad de localizar todos los textos del sistema en diferentes ficheros XML, uno por idioma soportado, de modo que a la hora de ejecutar la aplicación, carga el correspondiente al lenguaje del dispositivo. En caso de que no se disponga de uno, quedaría seleccionado el lenguaje establecido como estándar⁷. Para la aplicación descrita en este documento se consideran dos idiomas: inglés (como estándar) y español.

Una vez considerados los principios generales y los específicos para los elementos particulares de Android, la decisión de mayor relevancia en el diseño de IU en este sistema es la utilización de los llamados *Fragments* (o Fragmentos). Un Fragmento representa un comportamiento específico de una porción de la IU en una *Actividad*⁸; es decir, agrupa elementos de una interfaz (tanto el diseño gráfico como sus controles lógicos asociados) de modo que estos sean reutilizables entre diversos tamaños u orientaciones de pantalla.

En la figura 2.41 se aporta un esquema gráfico de la utilidad de estos elementos de diseño para una mejor comprensión.

De este modo, si bien el sistema se ha desarrollado para dispositivos móviles, la utilización de Fragmentos posibilita su adaptación a otros tamaños de pantalla de manera sencilla y simple.

Con todo, a modo de resumen, las decisiones o principios de diseño de IU adoptados son tres:

1. Las Reglas de Oro establecidas por Mandel.
2. Las normas de diseño Android para cada uno de sus componentes y elementos gráficos.
3. El empleo de Fragmentos para la agrupación y reutilización de elementos de IU entre diversos tamaños u operaciones de pantalla.

⁶La documentación puede ser consultada en <https://developer.android.com/design/index.html>, a partir de [AnDev].

⁷Para más detalles al respecto, consultar <https://developer.android.com/training/basics/supporting-devices/languages.html> a partir de [AnDev]

⁸Una Actividad es uno de los elementos básicos de cualquier aplicación Android. Sin entrar en detalles y por dar una primera aproximación, cada una de las pantallas o vistas de una aplicación suponen una Actividad. Más detalles en <https://developer.android.com/reference/android/app/Activity.html>, a partir de [AnDev].

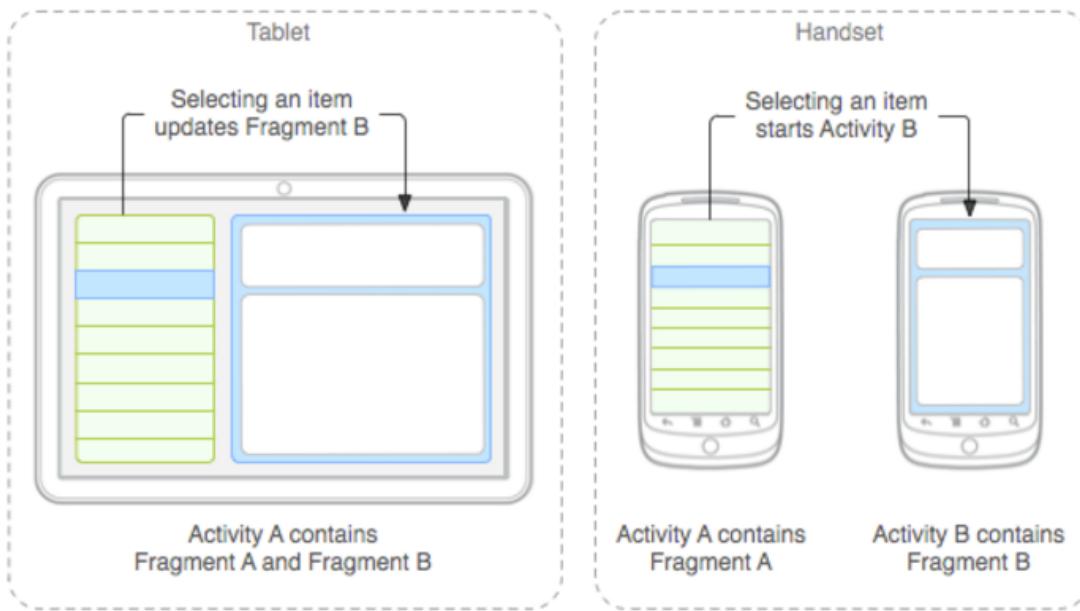


Figura 2.41: Fragmentos en el diseño de IUs. Fuente: [AnDev]

Estas decisiones han regido la construcción de la interfaz gráfica, proceso iterativo que consta de cuatro actividades estructurales [Man97]: 1) análisis y modelado, 2) diseño, 3) construcción y 4) validación.

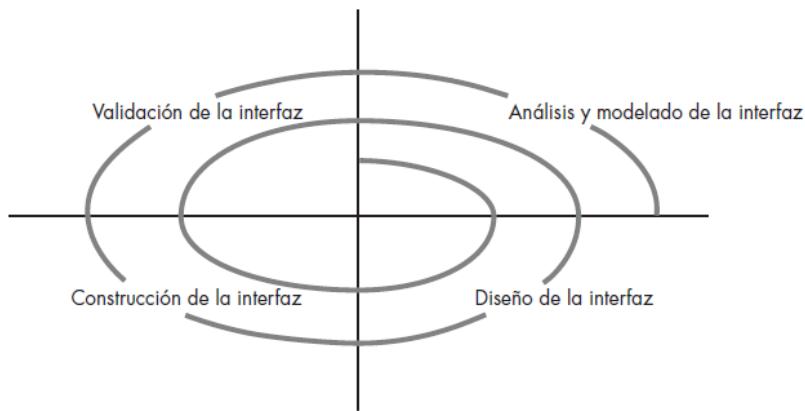


Figura 2.42: Proceso de diseño y construcción de una IU. Fuente: [Pre10]

El *análisis de la interfaz* se centra en el perfil de los usuarios que interactuarán con el sistema; es decir, se trabaja para entender la percepción del sistema para cada clase de usuarios.

La meta *del diseño de la interfaz* es definir un conjunto de objetos y acciones de ésta (y sus representaciones en la pantalla) que permitan al usuario efectuar todas las tareas definidas cumpliendo con los requisitos de usabilidad y diseño básicos.

La *construcción de la interfaz* comienza por lo general con la creación de un prototipo que permite evaluar los escenarios de uso.

La *validación de la interfaz* se centra en: 1) la capacidad de la interfaz para implementar correctamente todas las tareas y requerimientos del usuario; 2) el grado en el que la interfaz es fácil de usar y de aprender y 3) la aceptación que tiene por parte del usuario.

Finalmente, a continuación se presenta una relación de las pantallas fundamentales implementadas y de los requisitos funcionales que recoge cada una. Mencionar que este esquema no recoge todas las interfaces de usuario creadas, sino sólo aquellas que tienen implicación directa sobre alguno de los requisitos funcionales presentados en la sección 2.1. En cualquier caso, en el manual de usuario incluido en del apéndice C se puede observar la implementación y variedad final de las mismas.

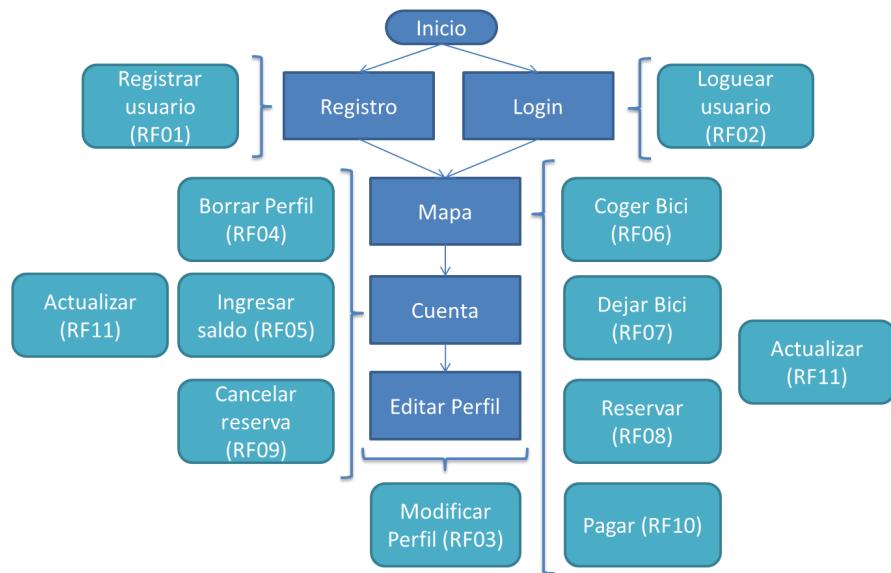


Figura 2.43: Relación de interfaces de usuario y requisitos funcionales

2.4. Implementación

2.4.1. Introducción

El modelo de implementación indica la distribución física de los artefactos software que componen el sistema, cuya arquitectura ha quedado modelada en el apartado anterior.

Con el objetivo de desarrollar adecuadamente este modelo, se consideran dos apartados:

1. **Herramientas utilizadas.** Se enumeran y comentan brevemente las tecnologías de mayor relevancia utilizadas en la implementación del software. Sin ahondar en detalles, se pretende aportar una aproximación a las herramientas seleccionadas con el objetivo de establecer una adecuada relación entre el software modelado anteriormente y el hardware que lo desarrollará y soportará.
2. **Distribución física.** Una vez comprendidas las tecnologías a utilizar, se muestra la distribución física de sistema, de modo que se tenga una visión global del mismo con los nodos involucrados y la comunicación entre ellos.

2.4.2. Herramientas utilizadas

Dada la arquitectura Cliente/Servidor que soporta el sistema, las herramientas y tecnologías se pueden agrupar en cada una de sus capas (cliente, servidor y base de datos) y en la comunicación entre ellas. Adicionalmente, se hará referencia al software controlador de versiones (CVS) empleado.

Este apartado no pretende profundizar en detalles técnicos, sino introducir las tecnologías de mayor relevancia presentes en el sistema y comentar las razones de su elección, de modo que se pueda comprender adecuadamente la relación entre la parte lógica y física de la aplicación.

Cliente

El cliente del sistema queda pensado, en su primera versión, para ser desplegado en dispositivos móviles con sistema operativo Android. El entorno de desarrollo para la programación de esta capa de datos es Android Studio.

- **Android.** Android es un sistema operativo basado en un núcleo Linux diseñado principalmente para dispositivos móviles con pantalla táctil. Se lanzó inicialmente en 2008 y su versión estable más actual es la 7, *Nougat*⁹.

⁹Tal y como se indicó en la sección de requisitos, Android denomina alfabéticamente y con nombres de postres o dulces a cada una de las versiones de su sistema operativo.

Las razones para la elección de este sistema, frente a otros como iOS o Windows, son principalmente la cobertura de mercado (en España, superior al 90 % en 2016¹⁰) y que el desarrollo de aplicaciones sobre esta plataforma es gratuito y basado en el lenguaje de programación Java, ampliamente extendido y con un fuerte soporte.

- **Android Studio.** Android Studio, basado en IntelliJ IDEA¹¹, es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android. Además del editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece más funciones para la mejora de la productividad durante la compilación de apps para Android¹².

Como se ha comentado, Android actualmente tiene desplegada la versión 7. Cada una de estas nuevas versiones incorpora nuevas funcionalidades y desestima o sustituye otras de versiones previas. Esto provoca que, a la hora de desarrollar una aplicación, se tenga que tener en cuenta el grado de compatibilidad de la misma con las versiones previas para lograr un alcance y cobertura razonables. Para ello, Android prevé la utilización de numerosas *librerías de compatibilidad*, que posibilitan la utilización de nuevos componentes o funcionalidades en versiones anteriores que, originalmente, no contaban con ellos. Del mismo modo, a la hora de crear una aplicación Android, se ha de dejar especificada la API mínima (y máxima) sobre la que dicha aplicación puede ser ejecutada (de modo que GooglePlay ofrezca aplicaciones adaptadas a la versión de cada dispositivo).

Para la aplicación desarrollada en este documento se ha seleccionado una compatibilidad hasta la versión 4.1 *Jelly Bean* (API 16), que alcanza una cobertura total del 97,5 % respecto al total de dispositivos repartidos por el mercado a nivel global¹³, tal y como se muestra en la figura 2.44.

Servidor

La capa intermedia del sistema queda soportado por el servidor de software libre GlassFish. El entorno de desarrollo empleado en este caso es NetBeans.

- **GlassFish.** GlassFish es un servidor basado en Apache Tomcat para el desarrollo y ejecución de aplicaciones regidas por la especificación *Java Platform*

¹⁰Dato a junio de 2016: http://cincodias.com/cincodias/2016/06/15/tecnologia/1465987235_750846.html

¹¹IDE para el desarrollo de sistemas informáticos desarrollado por JetBrains, más detalles en <https://www.jetbrains.com/idea/>.

¹²El detalle sobre estas funciones adicionales se puede consultar en <https://developer.android.com/studio/intro/index.html>, a partir de [AnDev].

¹³Dato a diciembre de 2016, Android actualiza esta información periódicamente en <https://developer.android.com/about/dashboards/index.html>, a partir de [AnDev].

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.2%
4.1.x	Jelly Bean	16	4.5%
4.2.x		17	6.4%
4.3		18	1.9%
4.4	KitKat	19	24.0%
5.0	Lollipop	21	10.8%
5.1		22	23.2%
6.0	Marshmallow	23	26.3%
7.0	Nougat	24	0.4%

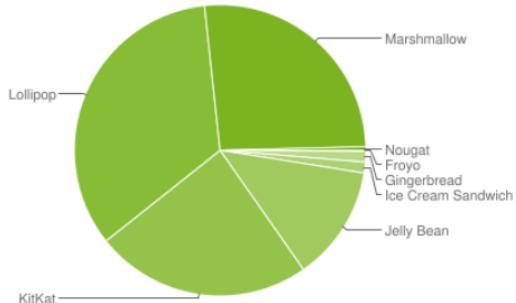


Figura 2.44: Datos oficiales Android de distribución de versiones a diciembre de 2016

Enterprise Edition (Java EE platform)¹⁴ y tecnologías web basadas en Java.

El motivo para la elección de este servidor es su carácter gratuito y de código libre; además del hecho de que, a diferencia de Tomcat, su instalación básica incluye un conjunto amplio de librerías, suficiente para cubrir las necesidades del sistema, sin que con ello se vea afectado su rendimiento de manera perceptible respecto a otras opciones más ligeras, como el citado Apache Tomcat.

- **NetBeans.** NetBeans es el IDE oficial para Java 8, supone un entorno de desarrollo de libre distribución y queda pensado principalmente para el lenguaje de programación Java.

La utilización de este entorno viene motivada principalmente por el soporte altamente desarrollado que tiene para la implementación de aplicaciones distribuidas, controlando desde el IDE las conexiones con servidor y base de datos y construyendo automáticamente el esqueleto de la aplicación a desplegar sobre dicho servidor (dependiendo de la naturaleza de la misma, como una basada en Web Services, por ejemplo).

¹⁴Plataforma de programación que incluye varias especificaciones para el desarrollo de aplicaciones distribuidas y empresariales, más detalles en [OraEE].

Base de datos

Capa de datos de la aplicación, implementada sobre un sistema MySQL y apoyado por el entorno MySQL Workbench.

- **MySQL Server.** MySQL es un sistema de gestión de bases de datos relacional y de código abierto. Tradicionalmente se ha enfocado a aplicaciones web donde la principal preocupación es la optimización de consultas sencillas, aspecto compartido por la aplicación en desarrollo.
- **MySQL Workbench.** MySQL Workbench es una herramienta visual de diseño de bases de datos que integra un conjunto de funcionalidades para facilitar la gestión de bases de datos MySQL.

Comunicación

Dado su carácter transversal, se ha decidido crear un grupo separado de tecnologías relativas a la comunicación entre capas de datos, si bien cada una puede quedar implementada en una u otra capa de la arquitectura.

- **Jackson.** Librería implementada en la capa cliente para la traducción de cadenas JSON a objetos Java y viceversa. El empleo de esta librería frente a otras es debido fundamentalmente a que suele dar mejores índices de rendimiento a medida que los ficheros a traducir van creciendo, aspecto potencialmente preferible¹⁵.
- **RESTful.** Servicios web basados en la arquitectura REST¹⁶ utilizados para el intercambio de datos entre el cliente y el servidor.

Se ha decidido utilizar este tipo de servicios web frente a otros como SOAP por varias razones:

- *Simpleza.* REST hace uso del protocolo estándar HTTP, lo que desemboca en una facilidad de desarrollo y entendimiento superior a SOAP. Ésto facilita, adicionalmente, su mantenimiento posterior. En general, hay pocas cosas que REST no haga de un modo más sencillo que SOAP.

¹⁵En Internet se pueden identificar numerosos estudios de rendimiento al respecto, todos con las mismas conclusiones mencionadas, se aporta uno de ellos: <https://dzone.com/articles/compare-json-api>.

¹⁶Arquitectura que, haciendo uso del protocolo HTTP, proporciona una API que utiliza cada uno de sus métodos (GET, POST, PUT, DELETE, etc.) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente. Una introducción teórica se puede encontrar en [WREST].

- *Amplitud de formatos.* REST permite multitud de formatos, mientras que SOAP sólo acepta XML. Aspecto de relevancia considerando, además, que JSON es más eficiente y sencillo que XML para el sistema en desarrollo, como ya se comentó en el apartado 2.3.3.
- *Rendimiento y escalabilidad.* REST presenta mejores métricas de rendimiento y escalabilidad respecto a SOAP.

Las características anteriores han hecho que REST se imponga a SOAP como servicio web dominante. De hecho, compañías como Yahoo o Google hacen uso de REST para todos sus servicios.

- **MySQL Connector.** Conejor estándar para la comunicación entre el servidor y la base de datos utilizando la API *Java Database Connectivity* (JDBC¹⁷).

Controlador de Versiones

Si bien no se trata de una herramienta directamente relacionada con el desarrollo de la aplicación, los controladores de versiones suponen un software imprescindible para cualquier desarrollo mínimamente complejo.

El software seleccionado ha sido *Git*, un CVS seleccionado por su eficiencia, sencillez y por su ampliación en *Github*, una plataforma colaborativa ampliamente extendida y utilizada¹⁸.

2.4.3. Distribución física

Una vez definidas las principales tecnologías utilizadas para la implementación del sistema, queda modelar la distribución física de los artefactos software presentes en el mismo con el objetivo de tener una visión general del soporte físico del sistema.

En la figura 2.45 se esquematiza la distribución mencionada. Se identifican tres nodos, distribución acorde a la arquitectura Cliente/Servidor utilizada:

- **Nodo Cliente.** Representa la capa de presentación del sistema, implementado en el dispositivo Android. Este nodo recoge la arquitectura de la aplicación modelada anteriormente en la figura 2.39 y es el utilizado por el usuario para interactuar. Los paquetes de clases utilizados para organizar el código son los siguientes:

¹⁷API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java utilizando el dialecto SQL, más detalles en [OraDB].

¹⁸En este documento no se busca entrar en más detalles sobre estas soluciones, para más detalles se puede consultar [Git] y [GitHb].

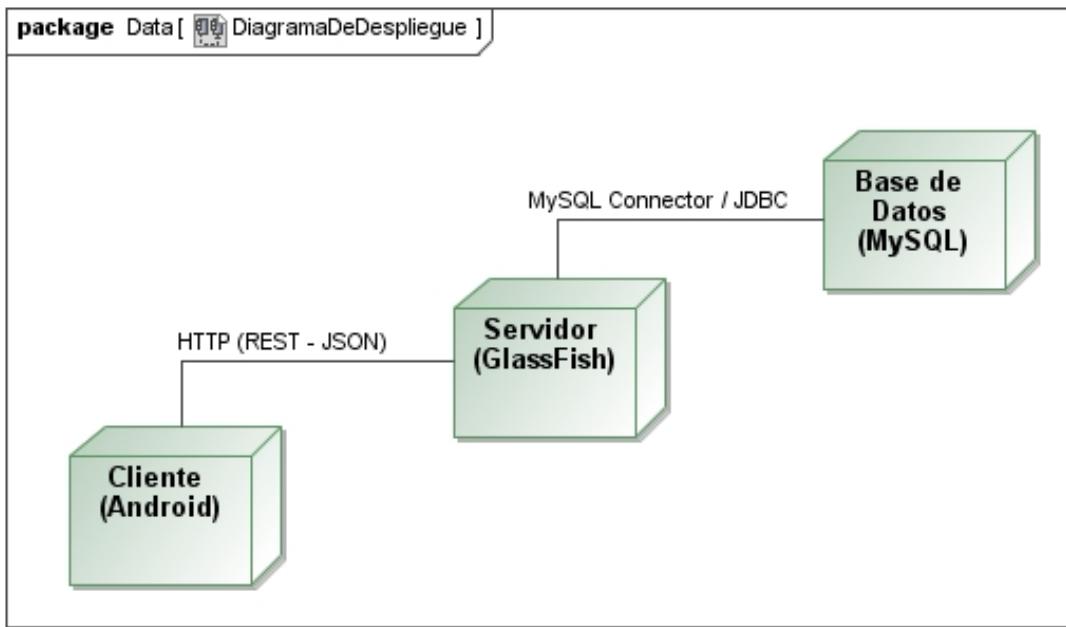


Figura 2.45: Diagrama de despliegue del sistema

1. **bikesmanager.** Paquete general donde se encuentran las clases que contienen la lógica de las interfaces de usuario (las *Activity* y sus *Fragment*).
 2. **bikesmanager.entity.** Paquete que contiene las clases de entidad de la aplicación.
 3. **bikesmanager.net.** Paquete que contiene las clases encargadas de la gestión de las conexiones HTTP con el servidor.
 4. **bikesmanager.util.** Paquete auxiliar con clases de apoyo o utilidad a algunas de las anteriores. Contiene, por ejemplo, adaptadores de listas desplegables.
- **Nodo Servidor.** La capa intermedia de la arquitectura, desplegada sobre GlassFish. Recoge la arquitectura del servidor modelada en la figura 2.40 y supone la capa que comunica la capa de presentación con la de datos. La comunicación con el cliente se realiza mediante el protocolo HTTP con los servicios web RESTful y el lenguaje JSON. Los paquetes de clases utilizados para organizar el código son los siguientes:
1. **services.** Paquete que contiene las clases encargadas de la gestión de los servicios web; es decir, las *Fachadas* del diagrama 2.40.

2. **entities.** Paquete que contiene las clases de entidad de la aplicación.
- **Nodo Base de Datos.** La capa de datos soportada por el sistema gestor MySQL. La comunicación con el servidor se realiza mediante los conectores estándar MySQL Connector y la API JDBC.

Capítulo 3

Pruebas

3.1. Introducción

Mediante la actividad conocida como *aseguramiento de la calidad del software*¹ se busca conseguir unos mínimos de calidad sobre las diversas etapas de ingeniería del software por las que pasa el producto: requerimientos, diseño e implementación (o código). Sin embargo, en este proceso pueden pasar errores inadvertidos que, sin unas pruebas adecuadamente planteadas y planificadas permanecerían en el sistema construido hasta su publicación.

Las pruebas representan, por tanto, la última oportunidad para valorar la calidad y, desde un punto de vista más práctico, descubrir errores. Sin embargo, citando a Pressman [Pre10], “no se puede probar la calidad. Si no está ahí antes de comenzar las pruebas, no estará cuando termine de probar”. Es decir, la calidad se ha de incorporar al software a lo largo de todo el proceso ingeniería, de modo que quede confirmada durante la etapa de pruebas.

Por tanto, el software se prueba para descubrir errores que se cometieron de manera inadvertida conforme se diseñó y construyó.

Las pruebas de software forman parte de un tema más amplio conocido como verificación y validación. La *verificación* se refiere al conjunto de tareas que garantizan que el software implementa correctamente una función específica. La *validación* es un conjunto diferente de tareas que aseguran que el software que se construye sigue los requerimientos establecidos en las etapas iniciales. La estrategia de pruebas, por tanto, se deberá enfocar para cubrir ambos aspectos: el sistema ha de funcionar adecuadamente de acuerdo a las funciones marcadas en la ERS (sección 2.1).

¹Para conocer más en detalle esta actividad se puede recurrir a [Pre10].

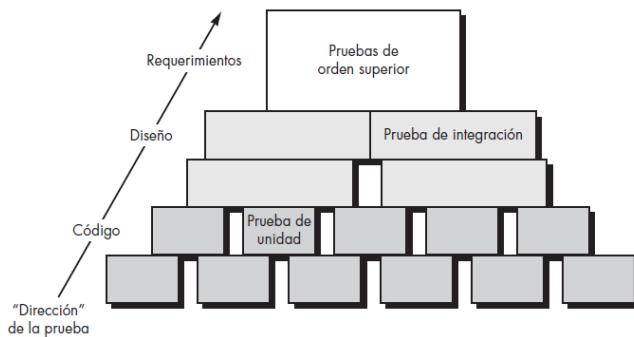


Figura 3.1: Pasos de las pruebas software *Fuente: [Pre10]*

3.2. Estrategia de pruebas

Una estrategia de pruebas software proporciona una guía que incorpora la planificación de la prueba, el diseño de casos de prueba, la ejecución de la prueba y la recolección y evaluación de los resultados.

Dentro del contexto de la Ingeniería del Software, las pruebas suponen una serie de cuatro pasos que se implementan de manera secuencial, tal y como se muestra en la figura 3.1.

Inicialmente, las pruebas se enfocan en cada componente de manera individual, son las *pruebas de unidad* y garantizan que todo módulo de software funciona adecuadamente como unidad, de ahí el nombre. A continuación, los componentes deben ensamblarse e integrarse para formar el paquete software completo. La *prueba de integración* aborda los conflictos asociados con los problemas de verificación y construcción de programas. Después de integrar (construir) el software, se realiza una serie de *pruebas de orden superior*, donde se identifican las *pruebas de validación*, que proporcionan la garantía final de que el software cumple con todos los requerimientos funcionales y de comportamiento abordados en la fase de requisitos, y las *pruebas del sistema*, para verificar que todos los elementos se mezclan de manera adecuada y que se logra el funcionamiento/rendimiento global del sistema deseado.

En este punto, cabe mencionar los atributos con que cuenta una “buena” prueba de software [Kan99]:

- *Una buena prueba tiene una alta probabilidad de encontrar un error.* Para ello, es conveniente que el examinador conozca el software para saber dónde podría fallar y probar dichos caminos.
- *Una buena prueba no es redundante.* Cada prueba ha debe tener un propósito diferente a otra.

- *Una buena prueba debe ser “la mejor de la camada”* [Kan99]. Para un conjunto de pruebas similares, se debe recurrir a la que tenga la mayor probabilidad de descubrir errores.
- *Una buena prueba no debe ser demasiado simple o demasiado compleja.* En general, cada prueba debe ejecutarse por separado.

Considerando el proceso descrito y las características mencionadas, en los apartados siguientes se aportarán los elementos más destacables empleados para probar la aplicación construida; si bien en este documento sólo se indicará el detalle concreto de las *pruebas de validación*, por ser aquellas que se centran en la funcionalidad específica del sistema y cuyo éxito depende del de las pruebas previas.

3.2.1. Pruebas de unidad

La *prueba de unidad* centra los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software, enfocándose en la lógica de procesamiento interno y de las estructuras de datos dentro de sus límites.

Las validaciones se han realizado siguiendo principalmente los métodos de *caja blanca*, filosofía de diseño de casos de prueba que [Pre10]: 1) garanticen que todas las rutas independientes dentro de un módulo se revisaron al menos una vez, 2) revisen todas las decisiones lógicas en sus lados verdadero y falso, 3) ejecuten todos los bucles en sus fronteras y dentro de sus fronteras operativas y 4) revisen estructuras de datos internas para garantizar su validez.

Sin entrar en detalles concretos, a continuación se mencionan las herramientas fundamentales utilizadas para realizar estas pruebas sobre los diferentes elementos del sistema.

Cliente - Android

La utilidad de mayor relevancia empleada para probar la aplicación Android de manera individual ha sido *Logcat*. Consiste en una herramienta de línea de comandos que vuelca un registro de mensajes del sistema, incluidos los seguimientos de pila, los casos de error del sistema y los mensajes escritos por el programador desde la app con la clase *Log*. Muestra los mensajes en tiempo real y mantiene un histórico para su consulta en caso de necesidad.

Con esta herramienta se puede hacer un seguimiento exhaustivo del estado de cada componente, estructura de datos o variable utilizada. Las posibilidades ofrecidas por la clase *Log* hacen que dicho seguimiento se pueda realizar de manera

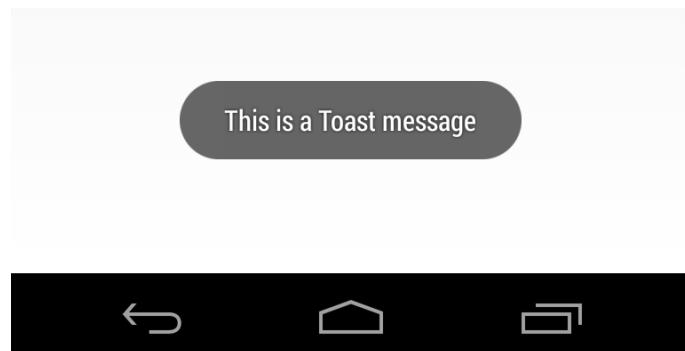


Figura 3.2: Ejemplo de un toast en Android

ordenada y sencilla².

Adicionalmente a Logcat, y desde un punto de vista de abstracción mayor, también se ha recurrido a la clase *Toast*. Un *toast* proporciona un *feedback* sencillo acerca de una operación mediante un mensaje de texto que se muestra durante unos instantes en la pantalla. Mientras que Logcat recurre a la línea de comandos del sistema, los *toast* se muestran en el propio dispositivo, siendo muy convenientes para mostrar al usuario el resultado de alguna operación que haya realizado³, en la figura 3.2 se aporta un ejemplo de uso de esta utilidad.

Servidor - GlassFish

El servidor queda programado en Java y como herramientas básicas de depuración se utilizaron las habituales escrituras en línea de comandos mediante la clase *System* y sus atributos *out* y *err*⁴, de modo que se pudiese hacer un seguimiento paso a paso del estado del elementos contenidos en esta capa de datos.

Adicionalmente, para la prueba de los diferentes módulos implementados en el servidor de manera aislada a la aplicación móvil, se ha recurrido a un complemento del navegador web Mozilla Firefox llamado *RESTClient*, un depurador para servicios web RESTful que permite realizar consultas tal y como las haría la aplicación. En la figura 3.3 se aporta una captura de pantalla del depurador.

Este tipo de prueba se podría considerar una prueba de mayor nivel al incluir también una conexión con la base de datos (la invocación de un servicio web

²Para más detalles acerca de Logcat se puede consultar <https://developer.android.com/studio/debug/am-logcat.html>, y acerca de la clase Log <https://developer.android.com/reference/android/util/Log.html>; ambos a partir de [AnDev].

³Como en la nota anterior, para más detalles acerca de Toast se puede consultar <https://developer.android.com/guide/topics/ui/notifiers/toasts.html>, a partir de [AnDev].

⁴Más detalles en <https://docs.oracle.com/javase/7/docs/api/java/lang/System.html>, a partir de [DocOr]

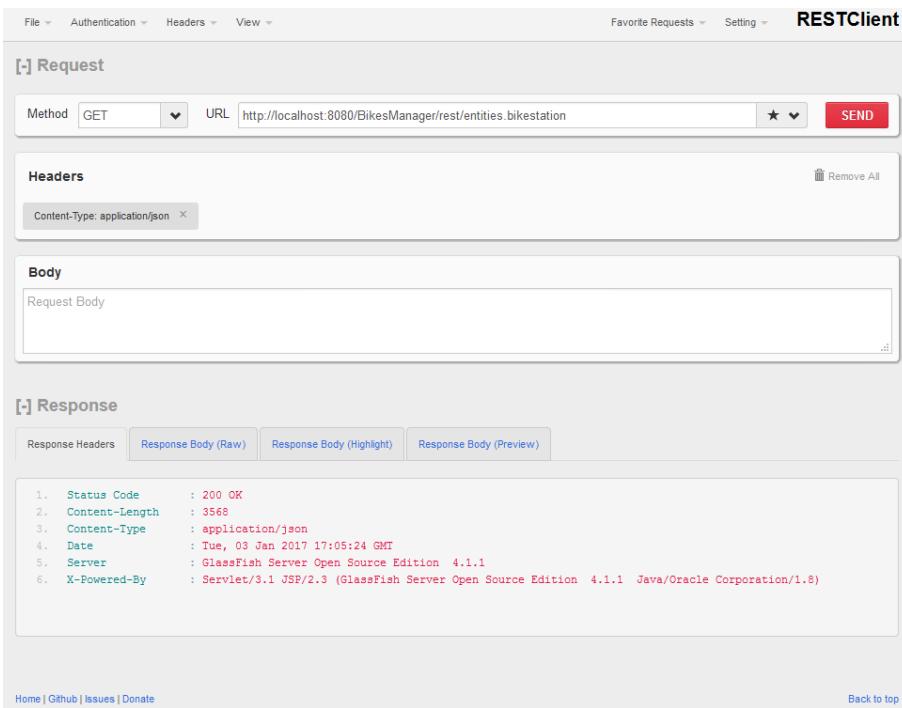


Figura 3.3: RESTClient, depurador de servicios web RESTful

finaliza mediante la obtención de un resultado al que se llega mediante la lectura o modificación de la base); sin embargo, considerando que el primer objetivo con el que se recurrió a esta herramienta fue el de ejercitarse, bajo un entorno controlado, los diferentes métodos del servidor (los servicios web individuales), se considera adecuado incluirla en esta sección.

Base de datos - MySQL

De manera individual, sobre la base de datos se han realizado consultas SQL directas mediante de la herramienta *MySQLWorkbench*, mencionada en la sección 2.4.2.

Cabe señalar que este tipo de pruebas sobre la capa de base de datos pueden no tener tanto interés como otras, puesto que dicha capa carece de lógica implementada para ser probada. El objetivo principal es el de ejercitarse la precisión y la integridad de los datos para asegurar que se almacenen, actualicen y recuperen de manera adecuada.

3.2.2. Pruebas de integración

El objetivo de las *pruebas de integración* es tomar los componentes probados de manera individual y construir la arquitectura señalada en la etapa de diseño. Dos han sido los enfoques fundamentales que se han seguido durante el desarrollo de este tipo de pruebas:

1. **Integración ascendente.** Enfoque principal, se comienza por la verificación de módulo atómicos y se van integrando y construyendo hacia niveles superiores.
2. **Pruebas de humo.** Enfoque secundario al anterior utilizado para agregados o arreglos no previstos, supone una integración constante según la cual el software se reconstruye (con el agregado de nuevos componentes) y se prueba día a día.

Los métodos para realizar las verificaciones se han basado principalmente en las técnicas de *caja negra*, enfocadas en los requerimientos funcionales del software. Es decir, las pruebas de caja negra intentan encontrar errores en las categorías siguientes [Pre10]: 1) funciones incorrectas o faltantes, 2) errores de interfaz, 3) errores en las estructuras de datos o en el acceso a bases de datos externas, 4) errores de comportamiento o rendimiento y 5) errores de inicialización y terminación.

Las herramientas utilizadas para completar este tipo de pruebas son las mismas que las empleadas en las pruebas de unidad, la diferencia radica en que este nivel tiene un enfoque más amplio, considerando los componentes y sus interrelaciones.

3.2.3. Pruebas de validación

Las *pruebas de validación* comienzan con la culminación de las pruebas de integración, enfocándose en las acciones visibles para el usuario y las salidas del sistema reconocibles por él. Como el resto de pruebas, la validación intenta descubrir errores, pero el enfoque se orienta en los requerimientos, siendo exitosa cuando el software funciona de forma que cumpla con las expectativas introducidas en la ERS.

Así, los casos de prueba se centran en la validación de los requisitos descritos en la sección 2.1. En los siguientes apartados se especifica el entorno de pruebas y el detalle para cada una.

Entorno de pruebas

El entorno de pruebas se ha basado en el parque de bicicletas de Madrid (Bicimad), se han creado una serie de estaciones directamente sobre la base de datos

con diferentes niveles de disponibilidad para la validación de diferentes casuísticas. Del mismo modo, se han creado una serie de usuarios con el objetivo de tener una aplicación más cercana a un entorno de ejecución real.

La batería de pruebas se ha ejecutado sobre varios dispositivos o plataformas diferentes:

- Emulador Android para PC, API 22 (Versión 5.1 - *Lollipop*).
- Dispositivo móvil, API 21 (Versión 5.0 - *Lollipop*).
- Dispositivo móvil, API 23 (Versión 6.0 - *Marshmallow*).

También se ha involucrado a varios usuarios con diferentes niveles de conocimiento en el ámbito tecnológico para evaluar la adaptación del sistema a cada uno. En este sentido, se han desarrollado tanto *pruebas alfa*, aquellas ejecutadas en un ambiente controlado y en presencia del desarrollador, como *pruebas beta*, ejecutadas sin presencia del desarrollador en un ambiente de “usuario final”.

Dado el entorno presentado, a continuación se describen las pruebas realizadas y sus resultados; si bien, por simplicidad, no se hará distinción entre dispositivos, usuarios o tipos de prueba, presentando cada una de manera única.

Especificación de pruebas

Se listan las pruebas planteadas por cada requisito funcional del sistema. Por cada una se describen las condiciones que se han de cumplir para considerar el test como superado.

▪ RF01: Registrar usuario

1. *Prueba 1. Registro satisfactorio.* Condiciones para superar la prueba:
 - El usuario se crea convenientemente en la base de datos.
 - El usuario tiene acceso a la funcionalidad de la aplicación una vez completado el registro.
2. *Prueba 2. Registro fallido. Nombre de usuario y/o dirección de correo electrónico en uso.* Condiciones para superar la prueba:
 - El usuario no se crea en la base de datos.
 - El usuario no tiene acceso a la funcionalidad de la aplicación.
 - La aplicación informa del error.
 - Se permite la modificación de los campos erróneos sin necesidad de salir del proceso.

3. *Prueba 3. Registro fallido. Ausencia de datos obligatorios.* Condiciones para superar la prueba:

- El usuario no se crea en la base de datos.
- El usuario no tiene acceso a la funcionalidad de la aplicación.
- La aplicación informa del error.
- Se permite la modificación de los campos erróneos sin necesidad de salir del proceso.

■ **RF02: Loguear usuario**

1. *Prueba 1. Autenticación satisfactoria.* Condiciones para superar la prueba:

- Los datos introducidos se corresponden con los almacenados en la base de datos.
- El usuario tiene acceso a la funcionalidad de la aplicación una vez completado el proceso.

2. *Prueba 2. Autenticación fallida. Ausencia de datos obligatorios.* Condiciones para superar la prueba:

- El usuario no tiene acceso a la funcionalidad de la aplicación.
- La aplicación informa del error.
- Se permite la modificación de los campos erróneos sin necesidad de salir del proceso.

3. *Prueba 3. Autenticación fallida. Nombre de usuario erróneo.* Condiciones para superar la prueba:

- El usuario no tiene acceso a la funcionalidad de la aplicación.
- La aplicación informa del error.

4. *Prueba 4. Autenticación fallida. Contraseña errónea.* Condiciones para superar la prueba:

- El usuario no tiene acceso a la funcionalidad de la aplicación.
- La aplicación informa del error.

■ **RF03: Modificar perfil**

1. *Prueba 1. Modificación satisfactoria.* Condiciones para superar la prueba:

- La aplicación confirma con el usuario la modificación.
- La aplicación confirma la modificación.

- El usuario se actualiza convenientemente, tanto en la base de datos como en la aplicación.
2. *Prueba 2. Modificación fallida. El nuevo nombre de usuario o correo electrónico (campos únicos) ya están en uso.* Condiciones para superar la prueba:
- El usuario no se modifica ni en la base de datos ni en la aplicación.
 - La aplicación informa del error.
 - Se permite la modificación de los campos erróneos sin necesidad de salir del proceso.
3. *Prueba 3. Modificación fallida. Ausencia de datos obligatorios.* Condiciones para superar la prueba:
- El usuario no se modifica ni en la base de datos ni en la aplicación.
 - La aplicación informa del error.
 - Se permite la modificación de los campos erróneos sin necesidad de salir del proceso.
4. *Prueba 4. Modificación fallida. El usuario cancela los cambios.* Condiciones para superar la prueba:
- La aplicación vuelve a su estado anterior sin cambios.

■ **RF04: Borrar perfil**

1. *Prueba 1. Borrado satisfactorio.* Condiciones para superar la prueba:
- La aplicación confirma con el usuario el borrado.
 - Las reservas activas del usuario se cancelan automáticamente.
 - La aplicación regresa a la pantalla de autenticación.
 - El usuario se elimina convenientemente, tanto en la base de datos como en la aplicación.
2. *Prueba 2. Borrado fallido. El usuario cancela los cambios.* Condiciones para superar la prueba:
- La aplicación vuelve a su estado anterior sin cambios.

■ **RF05: Ingresar saldo**

1. *Prueba 1. Ingreso satisfactorio.* Condiciones para superar la prueba:
- La aplicación informa al usuario de la realización de la operación.
 - El usuario se actualiza convenientemente, tanto en la base de datos como en la aplicación.

■ **RF06: Coger bicicleta**

1. *Prueba 1. Operación satisfactoria (sin reserva de bicicleta previa).* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - Se realiza el pago.
 - La aplicación informa al usuario de la realización de la operación.
 - El sistema se actualiza convenientemente, tanto en la base de datos como en la aplicación. Las entidades afectadas son: el Usuario y la Estación (donde entra la tarifa dependiente de la disponibilidad).
2. *Prueba 2. Operación satisfactoria (con reserva de bicicleta previa).* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - Se realiza el pago.
 - La aplicación informa al usuario de la realización de la operación.
 - El sistema se actualiza convenientemente, tanto en la base de datos como en la aplicación. Las entidades afectadas son: el Usuario, la Estación (donde entra la tarifa dependiente de la disponibilidad) y la Reserva (que se cancela automáticamente).
3. *Prueba 3. Operación errónea. No quedan bicicletas disponibles en la estación.* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa del error.
 - El sistema vuelve a su estado anterior sin cambios.
4. *Prueba 4. Operación errónea. El usuario no tiene saldo suficiente.* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa del error.
 - El sistema vuelve a su estado anterior sin cambios.
5. *Prueba 5. Operación errónea. El usuario ya tiene cogida una bici o una reserva de bicicletas en otra estación.* Condiciones para superar la prueba:

- La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa del error.
 - El sistema vuelve a su estado anterior sin cambios.
6. *Prueba 6. Intento de coger la última bicicleta por parte de dos usuarios al mismo tiempo.* Condiciones para superar la prueba:
- La aplicación concede la bicicleta al usuario que llegue el primero, informando al segundo del error y actualizando el sistema.

■ **RF07: Dejar bicicleta**

1. *Prueba 1. Operación satisfactoria (sin reserva de anclaje previa).* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa al usuario de la realización de la operación.
 - El sistema se actualiza convenientemente, tanto en la base de datos como en la aplicación. Las entidades afectadas son: el Usuario, la Estación (donde entra la tarifa dependiente de la disponibilidad).
2. *Prueba 2. Operación satisfactoria (con reserva de anclaje previa).* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa al usuario de la realización de la operación.
 - El sistema se actualiza convenientemente, tanto en la base de datos como en la aplicación. Las entidades afectadas son: el Usuario, la Estación (donde entra la tarifa dependiente de la disponibilidad) y la Reserva (que se cancela automáticamente).
3. *Prueba 3. Operación errónea. No quedan anclajes disponibles en la estación.* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa del error.
 - El sistema vuelve a su estado anterior sin cambios.
4. *Prueba 4. Operación errónea. El usuario no tiene cogida una bici o tiene una reserva de anclajes en otra estación.* Condiciones para superar la prueba:

- La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa del error.
 - El sistema vuelve a su estado anterior sin cambios.
5. *Prueba 5. Intento de dejar una bicicleta en el último anclaje por parte de dos usuarios al mismo tiempo.* Condiciones para superar la prueba:
- La aplicación concede el anclaje al usuario que llegue el primero, informando al segundo del error y actualizando el sistema.

■ RF08: Reservar

1. *Prueba 1. Operación satisfactoria (sin distinguir entre reservas de un tipo u otro).* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa al usuario de la realización de la operación.
 - La cuenta atrás de tiempo máximo de reserva (30 minutos) se inicia inmediatamente y se habilita la opción de cancelación.
 - El sistema se actualiza convenientemente, tanto en la base de datos como en la aplicación. Las entidades afectadas son: el Usuario, la Estación y la Reserva.
2. *Prueba 2. Operación fallida. El usuario trata de coger una segunda reserva del mismo tipo que una que ya tiene.* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa del error.
 - El sistema vuelve a su estado anterior sin cambios.
3. *Prueba 3. Operación fallida. El usuario trata de reservar una bici con otra cogida.* Condiciones para superar la prueba:
 - La aplicación informa al usuario del estado de la estación antes de que pueda completar la operación.
 - La aplicación informa del error.
 - El sistema vuelve a su estado anterior sin cambios.
4. *Prueba 4. Intento de reservar el último elemento de una estación (sin importar el tipo) por parte de dos usuarios diferentes al mismo tiempo.* Condiciones para superar la prueba:

- La aplicación concede la reserva al usuario que llegue el primero, informando al segundo del error y actualizando el sistema.

■ **RF09: Cancelar reserva**

1. *Prueba 1. Operación satisfactoria. Cancelación explícita (sin distinguir entre reservas de un tipo u otro).* Condiciones para superar la prueba:
 - La aplicación confirma con el usuario la cancelación.
 - La aplicación confirma la realización de la operación.
 - El sistema se actualiza convenientemente, tanto en la base de datos como en la aplicación. Las entidades afectadas son: el Usuario, la Estación y la Reserva.
2. *Prueba 2. Operación satisfactoria. Cancelación implícita (sin distinguir entre reservas de un tipo u otro).* Condiciones para superar la prueba:
 - Una vez superado los 30 minutos establecidos como tiempo máximo, la aplicación cancela automáticamente la reserva..
 - El sistema se actualiza convenientemente, tanto en la base de datos como en la aplicación. Las entidades afectadas son: el Usuario, la Estación y la Reserva.
3. *Prueba 3. Operación fallida. Cancelación explícita, el usuario no aprueba la operación.* Condiciones para superar la prueba:
 - El sistema vuelve a su estado anterior sin cambios.

Por su parte, las cancelaciones por eliminación de usuarios o por ejercer el derecho sobre las mismas ya se han tratado en pruebas anteriores.

■ **RF10: Pagar**

1. *Prueba 1. Pago satisfactorio.* Condiciones para superar la prueba:
 - El saldo disponible del usuario se actualiza convenientemente, tanto en la base de datos como en la aplicación.

Dado el carácter académico de la aplicación, el *pago* sólo implica la actualización del saldo disponible para el usuario dentro del sistema, con lo que no se considera la posibilidad de que éste no se realice correctamente.

■ **RF11: Actualizar**

1. *Prueba 1. Actualización satisfactoria.* Condiciones para superar la prueba:

- El sistema se actualiza convenientemente. Lo que, adicionalmente, implica que se cancelen automáticamente todas las reservas caducadas de otros usuarios (cancelación implícita a los 30 minutos), de modo que dichos recursos queden disponibles.
- La aplicación informa al usuario de la realización de la operación.

Los resultados de cada una de las pruebas anteriores y los comentarios asociados se pueden consultar en el apéndice B.

3.2.4. Pruebas de sistema

La *prueba del sistema* es una serie de diferentes pruebas cuyo propósito principal es ejercitar por completo, saliendo de la Ingeniería del Software y entrando en la de Sistemas, el sistema construido. Algunos de los ejemplos más representativos son [Pre10]:

- **Pruebas de recuperación**, dedicadas a la capacidad de relanzamiento (autónomo o con intervención humana) del software ante fallos que provoquen momentos de inactividad.
- **Pruebas de seguridad**, para asegurar el sistema ante ataques o intentos de entrada impropios.
- **Pruebas de esfuerzo**, destinadas a evaluar el comportamiento del sistema ante una demanda anormal de recursos.
- **Pruebas de rendimiento**, similares a las anteriores, estas pruebas están centradas en obtener unas métricas de funcionamiento mínimas.
- **Pruebas de despliegue o configuración**, para aquellos sistemas destinados a implementarse en varias plataformas, estas pruebas evalúan su desempeño en cada una de ellas.

Dado el tipo de pruebas encuadradas en este epígrafe, y considerando la naturaleza académica del proyecto presentado, las *pruebas de sistema* quedan fuera del alcance de la estrategia de pruebas para la aplicación construida. Algunas de las anteriores, sin embargo, sí se pueden asimilar a ciertas *pruebas de validación* antes mencionadas, como la evaluación del correcto funcionamiento en diferentes plataformas como *prueba de despliegue*, o el impedimento de acceso a un usuario no registrado como *prueba de seguridad*.

Capítulo 4

Conclusiones

El presente proyecto ha supuesto el desarrollo de una aplicación Android comunicada de manera remota con su servidor y base de datos para la gestión de parques públicos de bicicletas.

Sobre el sistema planteado inicialmente, se considera que se han cumplido los objetivos planteados en la sección 1.2. Además del básico referente a la construcción del sistema con sus funcionalidades principales, se ha hecho un estudio de las tecnologías involucradas para su selección fundamentada; el entendimiento de los estándares de diseño e implementación, en sus diferentes ámbitos, ha sido concienzudo; así como el desarrollo de una estrategia de pruebas adecuada, comenzando por las unidades más pequeñas de los elementos software y llegando al agregado que suponen los requisitos funcionales.

Dada la magnitud del proyecto, con una arquitectura Cliente/Servidor de tres capas, las tecnologías utilizadas han sido numerosas, desde las involucradas en las diferentes capas (cliente, servidor y base de datos), hasta aquellas requeridas para la comunicación entre ellas. Esto ha llevado a un aprendizaje y comprensión adecuada de lo que supone el despliegue de una aplicación completa.

El desarrollo sobre un sistema Android ha sido satisfactorio. Superadas las dificultades iniciales mediante el estudio y entendimiento adecuados de las particularidades del sistema, su construcción es relativamente ágil gracias a la documentación oficial publicada y a la comunidad de desarrolladores presentes en foros especializados. Asimismo, gracias a su predominancia en el mercado de los dispositivos móviles, por un lado, y las soluciones ideadas para soportar diversos dispositivos (lenguajes, tamaños de pantalla, versiones del sistema, etc.), por otro, las aplicaciones desarrolladas tienen un potencial de usuarios mayor que sobre otras plataformas.

Más allá de los aspectos técnicos, están aquellos referentes a la documentación y a la Ingeniería del Software. Mediante el apoyo de una adecuada bibliografía, se considera que el sistema ha quedado adecuadamente descrito y probado en sus

diferentes etapas.

Desde un punto de vista personal, la principal dificultad ha sido la falta de tiempo. Compatibilizar la realización del proyecto con un empleo especialmente absorbente desemboca en fines de semana de dedicación intensa para poder tener un grado de avance aceptable, seguidos de paradas de varios días hasta el fin de semana siguiente, reduciendo la continuidad y la productividad. Sin embargo, no deja de ser una dificultad que se supera mediante una adecuada mentalización y organización. Adicionalmente, quedan las dificultades habituales como son, por ejemplo, el desconocimiento de muchas de las tecnologías requeridas. Si bien aspectos como este terminan en un aporte de conocimientos técnicos y organizativos intenso y enriquecedor.

4.1. Líneas futuras

La aplicación supone una base tecnológica para el desarrollo y mejora de los servicios de gestión de parques públicos de bicicletas. Se consideran las siguientes posibles líneas futuras de desarrollo y estudio:

- Desarrollo de la adaptación de precios a la hora de coger, dejar o reservar bicicletas, dependiendo de la disponibilidad de cada estación. Actualmente la aplicación duplica el precio para coger una bicicleta si la disponibilidad de la estación cae por debajo del 50 %.
- Desarrollo de un sistema de alertas o notificaciones automático que avise al usuario de la disponibilidad de bicicletas o anclajes en una estación elegida por él. De este modo el usuario podría seleccionar una serie de estaciones “favoritas” sobre las que periódicamente se informe de la disponibilidad o, en caso de que haya querido reservar y no hubiese sido posible, alerta cuando haya una bici preparada.
- Inclusión de un mayor volumen de estados para las bicicletas. En la aplicación actual, una bici puede estar disponible, reservada o cogida; se podría incluir un nuevo estado que hiciese referencia a, por ejemplo, las bicicletas averiadas, de modo que un usuario que identificase una avería en una bici o estación informase mediante una breve descripción a través de la aplicación.
- Desarrollo de versiones sobre diferentes plataformas. Actualmente la aplicación se encuentra desarrollada únicamente para dispositivos Android, se podría realizar una adaptación de IU para tablets, desarrollo para otros sistemas operativos, como iOS, o plataformas, como una versión web.

- Relacionado con el punto anterior, la aplicación Android se debería mantener actualizada y adaptada las nuevas versiones o funcionalidades que se vayan publicando.
- Extender la batería de pruebas para llegar a cubrir las *pruebas de sistema* fundamentales, extensión que conllevaría una adaptación del sistema diseñado de cara a superarlas y dando lugar a un entorno más consistente y profesional.

Apéndice A

API implementada

En este anexo se aporta el detalle de la API implementada en el servidor de aplicación tal y como se introdujo en la sección 2.3.3, referida al diseño de interfaces.

El detalle aporta, por método, su ruta y objetivo concreto; así como los parámetros de entrada, salida y ejemplos asociados. Señalar que todas las salidas contempladas se describen suponiendo un mensaje de respuesta HTTP 200 (ejecución correcta del método).

Cabe recordar que las URLs se construyen a partir de la plantilla

http : // [IP_SERV] : [PORT] / BikesManager / rest / entities . [ENTITY]

(ver sección referenciada anteriormente para más detalles), sobre la que se construyen las llamadas particulares a los métodos mediante parámetros adicionales sobre la URL, tal y como queda descrito a continuación.

Finalmente, apuntar que para conocer el formato de las cadenas JSON a introducir u obtener, basta con ejecutar cualquiera de los métodos GET aportados a continuación directamente sobre un navegador (y referenciando a una base de datos con contenido).

■ Estación

ID	Método	Ruta	Objetivo	Entrada	Salida	Ejemplo	Comentarios
1	POST	-	Método básico para la creación de entidades de tipo Estación en base de datos. Este método, si bien está disponible, no se ha utilizado en la aplicación desarrollada puesto que las estaciones se crean directamente en base de datos.	-	-	<code>http://localhost:8080/BikesManager/rest/entities/bikestation</code>	La petición ha de venir acompañada de una cadena JSON con la entidad a actualizar. Este método supone la actualización básica de la entidad, sin necesidad de realizar comprobaciones previas para ello, con lo que siempre devolverá SERVER.OK
2	PUT	{id}	Método básico para la actualización directa (sin comprobaciones) de entidades de tipo Estación en la base de datos.	id: Integer con el ID de la estación a actualizar.	JSON: mensaje entity-bikestation- -SERVER.OK.	<code>http://localhost:8080/BikesManager/rest/entities.bikestation/1</code>	La petición ha de venir acompañada de una cadena JSON con la entidad a actualizar. Este método supone la actualización básica de la entidad, sin necesidad de realizar comprobaciones previas para ello, con lo que siempre devolverá SERVER.OK
3	PUT	{operation} / {id}	Método para la actualización de Estaciones cuando se realiza alguna de las siguientes operaciones: coger bicicleta, dejar bicicleta y gestionar reservas de bicicleta o anclajes. Este método se ha implementado adicionalmente al anterior puesto que las operaciones mencionadas requieren de comprobaciones sobre el servidor para evaluar la disponibilidad de cada estación y ofrecer al usuario un resultado consistente. El método descarga de la base de datos la estación sobre la que se quiere operar y la actualiza en caso de ser posible.	operation: String con la operación a realizar sobre la estación (<i>take</i> , <i>book_bike</i> , <i>book_slots</i> , <i>take_book</i> , <i>leave_book</i> , <i>cancel_book</i> , <i>bikestation_on</i>); id: Integer con el ID de la estación a actualizar.	JSON: mensaje entity-bikestation- -SERVER.OK o entity_bikestation- -SERVER.KO.	<code>http://localhost:8080/BikesManager/rest/entities.bikestation/take/1</code>	La petición ha de venir acompañada de una cadena JSON con la entidad a actualizar. Este método devuelve SERVER.OK si la operación ha sido posible basándose en condiciones de disponibilidad de la estación y SERVER.KO en caso contrario.
4	DELETE	{id}	Método básico para el borrado de entidades de tipo Estación de la base de datos. Este método no se ha utilizado en la aplicación desarrollada puesto que las estaciones se eliminan directamente en base de datos.	id: Integer con el ID de la estación a eliminar.	-	<code>http://localhost:8080/BikesManager/rest/entities.bikestation/1</code>	-
5	GET	{id}	Método básico para obtener entidades de tipo Estación de la base de datos a partir de su ID. Este método no se ha utilizado en la aplicación desarrollada puesto que las búsquedas individuales se realizan sobre la dirección, no sobre el ID.	id: Integer con el ID de la estación a buscar.	JSON: estación buscada.	<code>http://localhost:8080/BikesManager/rest/entities.bikestation/1</code>	-
6	GET	stationAddress / {address}	Método adicional al anterior para obtener entidades de tipo Estación de la base de datos a partir de su dirección (campo único).	address: String con la dirección de la estación a buscar.	JSON: estación buscada.	<code>http://localhost:8080/BikesManager/rest/entities.bikestation/stationAddress/Plaza_de_la_Puerta_del_Sol</code>	Los elementos de la dirección han de venir separados por “_” y no por espacios en blanco en la URL.

Continúa en la página siguiente

Tabla A.1 – Continúa desde la página anterior

ID	Método	Ruta	Objetivo	Entrada	Salida	Ejemplo	Comentarios
7	GET	-	Método básico para la obtención del listado completo de entidades de tipo Estación creadas en la base de datos.	-	JSON: listado completo de estaciones.	<code>http://localhost:8080/BikesManager/rest/entities.bikestation</code>	Este método, antes de devolver el listado de estaciones, actualiza las posibles reservas cadaudas para dar salida a un listado consistente.
8	GET	{from} / {to}	Método básico para la obtención de un listado acotado de entidades de tipo Estación creadas en la base de datos. Se obtienen tantas entidades como las indicadas en el parámetro <i>to</i> a partir de la estación con el ID más cercano al indicado en el parámetro <i>from</i> . Este método no se ha utilizado en la aplicación desarrollada.	from: Integer con el ID de la estación desde la que se parte para la búsqueda; to: Integer con el número de estaciones a obtener.	JSON: rango de estaciones buscado.	<code>http://localhost:8080/BikesManager/rest/entities.bikestation/1/5</code>	-
9	GET	count	Método básico para conocer el número de entidades de tipo Estación creadas en la base de datos. Este método no se ha utilizado en la aplicación desarrollada.	-	PLAIN: número de entidades creadas en la base de datos.	<code>http://localhost:8080/BikesManager/rest/entities.bikestation/count</code>	-

Tabla A.1: API para la entidad Estación

■ Usuario

ID	Método	Ruta	Objetivo	Entrada	Salida	Ejemplo	Comentarios
1	POST	-	Método básico para la creación de entidades de tipo Usuario en la base de datos. Este método comprueba la disponibilidad del nombre de usuario o dirección de correo elegidas, informando de si la creación ha sido posible o no.	-	JSON: mensaje entity_bikeuser-SERVER.OK o entity_bikeuser-SERVER.KO.	<code>http://localhost:8080/BikesManager/rest/entities.bikeuser</code>	La petición ha de venir acompañada de una cadena JSON con la entidad a crear. Este método devuelve SERVER.OK (crea el usuario) si no hay coincidencias de nombre de usuario y/o correo electrónico o SERVER.KO (no se crea el usuario) en caso contrario.
2	PUT	{id}	Método básico para la actualización directa (sin comprobaciones) de entidades de tipo Usuario en la base de datos.	id: Integer con el ID del usuario a actualizar.	JSON: mensaje entity_bikeuser-SERVER.OK.	<code>http://localhost:8080/BikesManager/rest/entities.bikeuser/1</code>	La petición ha de venir acompañada de una cadena JSON con la entidad a actualizar. Este método supone la actualización básica de la entidad, sin necesidad de realizar comprobaciones previas para ello, con lo que siempre devolverá SERVER.OK una vez ejecutada.

Continúa en la página siguiente

Tabla A.2 – Continúa desde la página anterior

ID	Método	Ruta	Objetivo	Entrada	Salida	Ejemplo	Comentarios
3	PUT	basicdata / {id}	Método adicional al anterior para actualizar Usuarios cuando se quiere modificar el nombre de usuario o dirección de correo, modificación que requieren de comprobaciones de disponibilidad adicionales.	id: Integer con el ID del usuario a actualizar.	JSON: mensaje entity_bikeuser-SERVER.OK o entity_bikeuser-SERVER.KO.	http://localhost:8080/BikeManager/rest/entities.bikeuser/basicdata/1	La petición ha de venir acompañada de una cadena JSON con la entidad a actualizar. Este método devuelve SERVER.OK o SERVER.KO dependiendo de la disponibilidad de los nuevos datos introducidos.
4	DELETE	{id}	Método básico para el borrado de entidades de tipo Usuario de la base de datos.	id: Integer con el ID del usuario a eliminar.	JSON: mensaje entity_bikeuser-SERVER.OK.	http://localhost:8080/BikeManager/rest/entities.bikeuser/1	Dado que la petición parte del propio usuario que quiere eliminar su cuenta, no hay posibilidad de error, con lo que el método devuelve siempre SERVER.OK.
5	GET	{id}	Método básico para obtener usuarios de la base de datos a partir de su ID. Este método no se ha utilizado en la aplicación desarrollada, puesto que las búsquedas individuales se realizan sobre el nombre de usuario, no sobre el ID.	id: Integer con el ID del usuario a buscar.	JSON: usuario buscado.	http://localhost:8080/BikeManager/rest/entities.bikeuser/1	
6	GET	user / {username}	Método adicional al anterior para obtener datos de la base de datos a partir del nombre de usuario.	username: String con el nombre de usuario a buscar.	JSON: usuario buscado.	http://localhost:8080/BikeManager/rest/entities.bikeuser/user/1	
7	GET	-	Método básico para la obtención del listado completo de entidades de tipo Usuario creadas en la base de datos. No es un método utilizado en la aplicación desarrollada puesto que sólo se realizan consultas de usuarios individuales.	-	JSON: listado completo de usuarios.	http://localhost:8080/BikeManager/rest/entities.bikeuser	
8	GET	{from} / {to}	Método básico para la obtención de un listado acotado de entidades de tipo Usuario creadas en la base de datos. Se obtienen tantas entidades como las indicadas en el parámetro <i>to</i> a partir de la estación con el ID más cercano al indicado en el parámetro <i>from</i> . Este método no se ha utilizado en la aplicación desarrollada.	from: Integer con el ID del usuario desde el que se parte para la búsqueda; to: Integer con el número de usuarios a obtener.	JSON: rango de usuarios buscado.	http://localhost:8080/BikeManager/rest/entities.bikeuser/1/5	
9	GET	count	Método básico para conocer el número de entidades de datos. Este método no se ha utilizado en la aplicación desarrollada.	-	PLAIN: número de entidades creadas en la base de datos.	http://localhost:8080/BikeManager/rest/entities.bikeuser/count	

Tabla A.2: API para la entidad Usuario

■ Reserva

ID	Método	Ruta	Objetivo	Entrada	Salida	Ejemplo	Comentarios
1	POST	-	Método básico para la creación de entidades de tipo Reserva en la base de datos.	-	JSON: mensaje entity-booking-<SERVER.OK.	http://localhost:8080/BikesManager/rest/entities.booking	La petición ha de venir acompañada de una cadena JSON con la entidad a crear. Este método se ejecuta únicamente cuando la reserva puede ser verdaderamente creada por condiciones de disponibilidad, con lo que siempre devuelve SERVER.OK.
2	PUT	{id}	Método básico para la actualización directa (sin comprobaciones) de entidades de tipo Reserva en la base de datos. Este método, si bien queda disponible, no se ha utilizado en la aplicación desarrollada puesto que las reservas sólo se pueden crear o cancelar (eliminar).	id: Integer con el ID de la reserva a actualizar.	-	http://localhost:8080/BikesManager/rest/entities.booking/1	La petición ha de venir acompañada de una cadena JSON con la entidad a actualizar.
3	DELETE	{id}	Método básico para el borrado de entidades de tipo Reserva en la base de datos. Este método, si bien queda disponible, no se ha utilizado, puesto que para el borrado de reservas se ha implementado el método descrito a continuación.	id: Integer con el ID de la reserva a eliminar.	-	http://localhost:8080/BikesManager/rest/entities.booking/1	-
4	DELETE	{username}/booking-type	Método adicional al anterior para el borrado de entidades de tipo Reserva de la base de datos. Puesto que un usuario sólo puede cancelar sus propias reservas, este método realiza, bajo petición, una búsqueda del tipo de reserva especificado a su nombre y la elimina.	username: String con el nombre de usuario que solicita la eliminación de la reserva; bookingType: Integer con el tipo de reserva a cancelar (1 para las bicis y 2 para los anclajes),	JSON: mensaje entity-booking-<SERVER.OK.	http://localhost:8080/BikesManager/rest/entities.booking/usuario/1	Este método lo ejecuta el usuario sobre sus propias reservas, no hay posibilidad de error, devolviendo siempre SERVER.OK.
5	GET	{id}	Método básico para obtener reservas de la base de datos a partir de su ID. Este método no se ha utilizado en la aplicación desarrollada.	id: Integer con el ID de la reserva a buscar.	JSON: reserva buscado.	http://localhost:8080/BikesManager/rest/entities.booking/1	-
6	GET	-	Método básico para la obtención del listado completo de entidades de tipo Reserva creadas en la base de datos. Este método no se ha utilizado en la aplicación desarrollada.	-	JSON: listado completo de reservas.	http://localhost:8080/BikesManager/rest/entities.booking	-

Continúa en la página siguiente.

Tabla A.3 – Continúa desde la página anterior

ID	Método	Ruta	Objetivo	Entrada	Salida	Ejemplo	Comentarios
7	GET	{from} / {to}	Método básico para la obtención de un listado acotado de entidades de tipo Usuario creadas en la base de datos. Se obtienen tantas entidades como las indicadas en el parámetro <i>to</i> a partir de la estación con el ID más cercano al indicado en el parámetro <i>from</i> . Este método no se ha utilizado en la aplicación desarrollada.	from: Integer con el ID de la reserva desde la que se parte para la búsqueda; to: Integer con el número de reservas a obtener.	JSON: rango de reservas buscado.	http://localhost:8080/Bikesanager/rest/entities.booking/1/5	-
8	GET	count	Método básico para conocer el número de entidades de tipo Estación creadas en la base de datos. Este método no se ha utilizado en la aplicación desarrollada.	-	PLAIN: número de entidades creadas en la base de datos.	http://localhost:8080/Bikesanager/rest/entities.booking/count	-

Tabla A.3: API para la entidad Reserva

Apéndice B

Resultados de las pruebas de validación

Se aportan los resultado de las pruebas planteadas en la sección 3.2.3. Adicionalmente, se incluyen los posibles ajustes surgidos a partir de la realización de las mismas, si bien los resultados mostrados en este apéndice hacen referencia a la última versión de la aplicación, sobre la cual no se derivaron más modificaciones.

RF	Prueba	Resultado	Comentarios
RF01	1	✓	Se añade una pantalla una vez se completa el registro a modo de confirmación explícita. Además, otorga 5€ al nuevo usuario para que pueda empezar a operar inmediatamente.
	2	✓	Prueba completada satisfactoriamente.
	3	✓	Prueba completada satisfactoriamente.
RF02	1	✓	Prueba completada satisfactoriamente.
	2	✓	Se inhabilita el botón de acceso hasta que se hayan completado todos los campos para reducir la generación de diálogos.
	3	✓	Prueba completada satisfactoriamente.
	4	✓	Prueba completada satisfactoriamente.
RF03	1	✓	Prueba completada satisfactoriamente.
	2	✓	Prueba completada satisfactoriamente.
	3	✓	Prueba completada satisfactoriamente.
	4	✓	Prueba completada satisfactoriamente.
RF04	1	✓	Se introduce la restricción referente a que si el usuario tiene una bici cogida no pueda borrar su perfil hasta que no la devuelva.

Continúa en la página siguiente

Tabla B.1 – Continúa desde la página anterior

RF	Prueba	Resultado	Comentarios
	2	✓	Prueba completada satisfactoriamente.
RF05	1	✓	Prueba completada satisfactoriamente.
RF06	1	✓	Dos mejoras de IU: 1) Además de la confirmación inmediata, se crea una barra de estado que informa permanentemente si el usuario tiene bici o no; y 2) se introduce un código de colores sobre las estaciones para informar visualmente de su disponibilidad.
	2	✓	Prueba completada satisfactoriamente (aplica el comentario realizado en la prueba RF06 - 1).
	3	✓	Prueba completada satisfactoriamente.
	4	✓	En la barra de estado comentada anteriormente se introduce también el saldo disponible y un acceso directo a la modificación del perfil para contar con un acceso más ágil al ingreso de dinero.
	5	✓	Prueba completada satisfactoriamente.
	6	✓	Prueba completada satisfactoriamente.
RF07	1	✓	Prueba completada satisfactoriamente (aplica el comentario realizado en la prueba RF06 - 1).
	2	✓	Prueba completada satisfactoriamente (aplica el comentario realizado en la prueba RF06 - 1).
	3	✓	Prueba completada satisfactoriamente.
	4	✓	Prueba completada satisfactoriamente.
	5	✓	Prueba completada satisfactoriamente.
RF08	1	✓	Se introduce un código de colores sobre las estaciones para informar visualmente de la estación sobre la que se ha efectuado la reserva. Este código desaparece si no hay reservas activas.
	2	✓	Las opciones de reserva que no se puedan realizar se desactivan para reducir las necesidades de navegación del usuario y ofrecer una experiencia más ágil.

Continúa en la página siguiente

Tabla B.1 – Continúa desde la página anterior

RF	Prueba	Resultado	Comentarios
	3	✓	Prueba completada satisfactoriamente (aplica el comentario de la prueba RF08 - 2).
	4	✓	Prueba completada satisfactoriamente.
RF09	1	✓	Prueba completada satisfactoriamente (aplica el comentario realizado en la prueba RF08 - 1)
	2	✓	Prueba completada satisfactoriamente (aplica el comentario realizado en la prueba RF08 - 1)
	3	✓	Prueba completada satisfactoriamente (aplica el comentario realizado en la prueba RF08 - 1)
RF10	1	✓	Prueba completada satisfactoriamente.
RF11	1	✓	Para una visualización general más completa del estado del sistema, se introducen dos vistas adicionales: una lista con el detalle de todas las estaciones y unos gráficos de disponibilidad.

Tabla B.1: Resultados de las pruebas de validación

Apéndice C

Manual de usuario

Configuración inicial y pantalla inicial

Si se acaba de instalar la aplicación, se solicitará automáticamente la dirección del servidor para poder conectarse (esta información puede modificarse posteriormente):

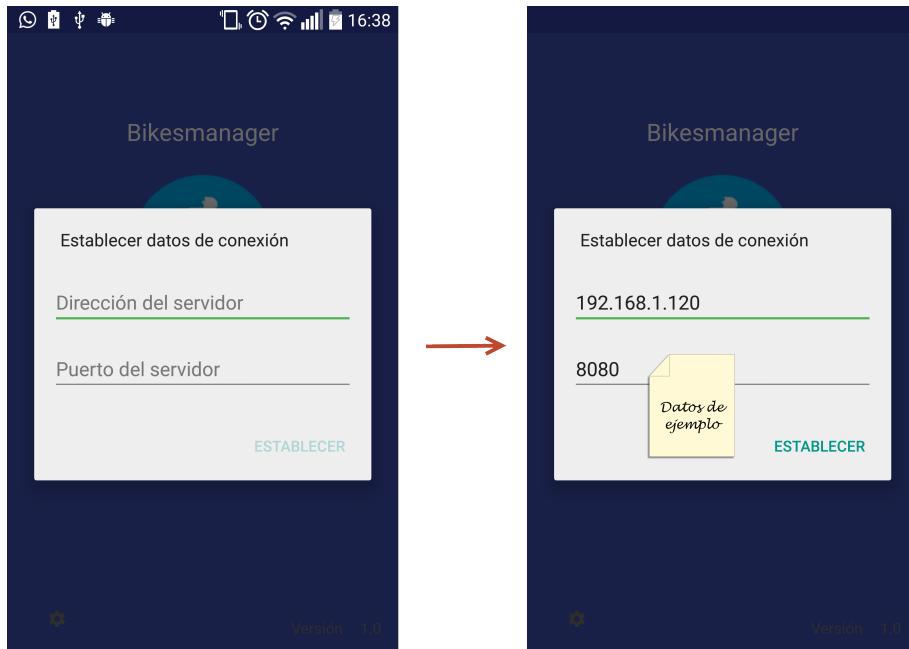


Figura C.1: Configuración inicial de la conexión

A partir de este punto, se tiene acceso a la pantalla inicial de la aplicación, desde la que se podrá registrar un nuevo usuario o iniciar sesión con uno existente.

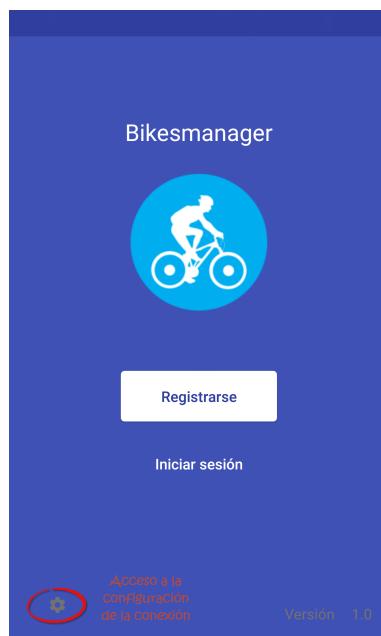


Figura C.2: Pantalla inicial de la aplicación

Registro de usuarios

Mediante el botón “Registrarse” se accede al registro de nuevos usuarios, donde se solicitarán una serie de datos de obligada cumplimentación. Una vez completo el registro, se recibe un pequeño regalo...☺:

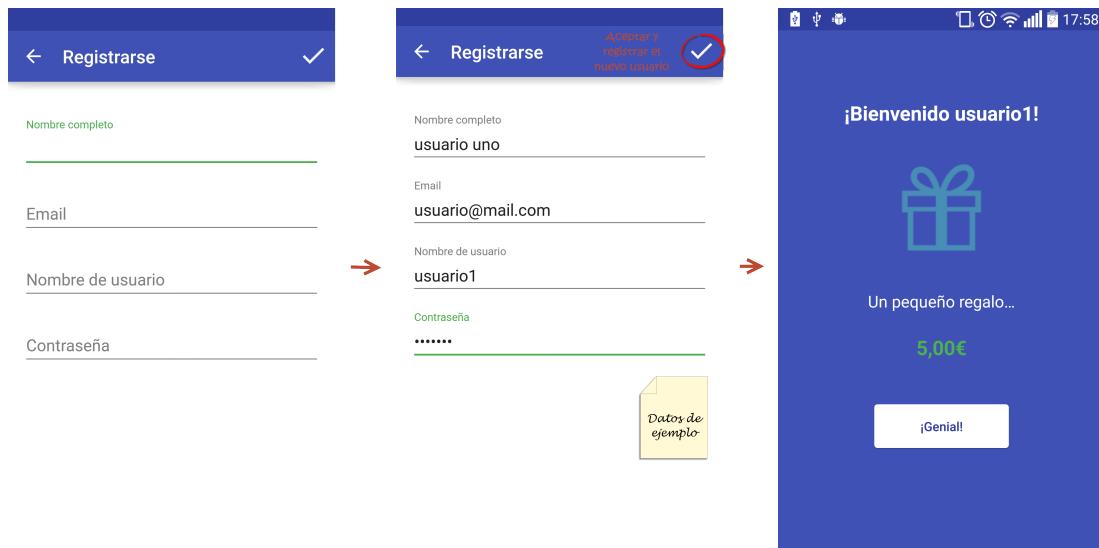


Figura C.3: Registro de nuevos usuarios

A partir de este punto, se tiene acceso a operar con la aplicación.

Inicio de sesión

Mediante el botón “Iniciar Sesión” los usuarios registrados pueden acceder a la aplicación, se solicitará el nombre de usuario y la contraseña seleccionados en la etapa de registro:

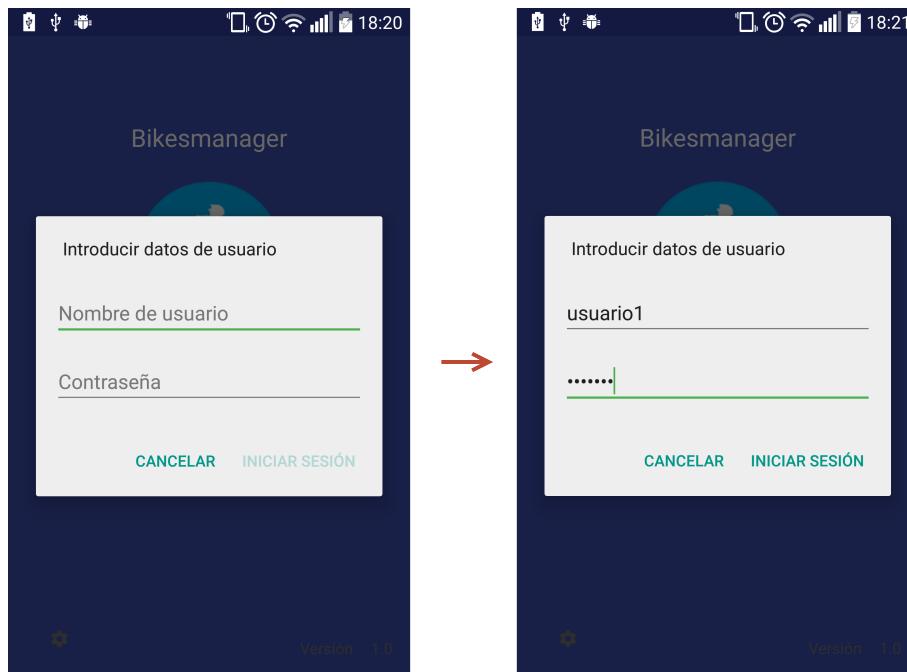


Figura C.4: Inicio de sesión

Pantalla principal de la aplicación: vistas y opciones adicionales

Una vez iniciada sesión, se accede a la pantalla principal de la aplicación: el mapa con las estaciones disponibles, que presentan cuatro códigos de colores posibles:

- **Rojo.** No quedan bicicletas disponibles en la estación.
- **Amarillo.** Quedan menos del 50 % de bicicletas disponibles en la estación. En las estaciones con este nivel de disponibilidad la tarifa es el doble de lo habitual.
- **Verde.** Quedan más del 50 % de bicicletas disponibles en la estación.
- **Azul.** El usuario tiene una reserva en esta estación.

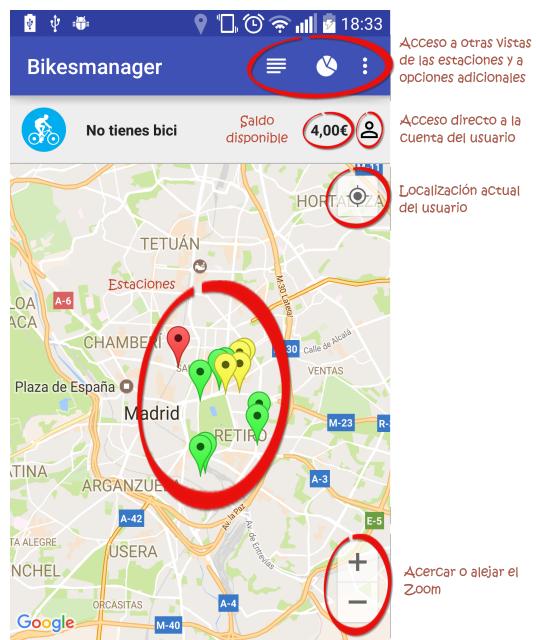


Figura C.5: Pantalla principal de la aplicación

El estado concreto de una estación (bicis y anclajes disponibles y tarifa actual) se puede consultar mediante su selección:

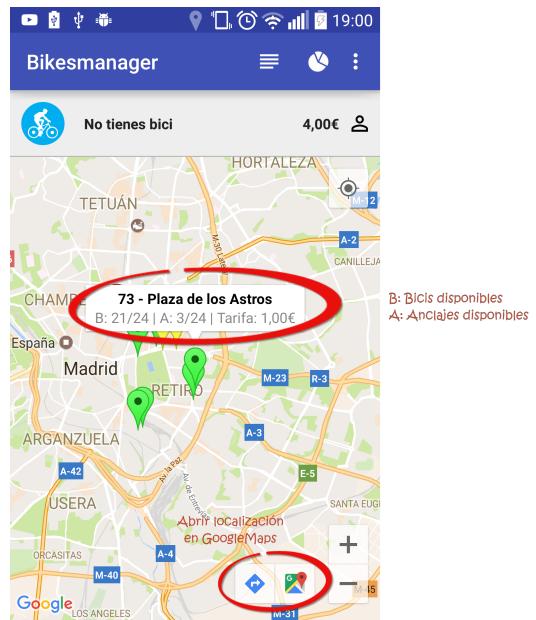


Figura C.6: Detalle de una estación

Mediante los botones superiores se tiene acceso a otras vistas del parque de bicicletas y a opciones adicionales:



Figura C.7: Diferentes vistas y opciones

El primero ofrece un listado desplegable con el detalle de cada estación. Aporta una vista diferente del conjunto de bicicletas que permite una búsqueda más sencilla y una visión más completa y detallada de cada una.

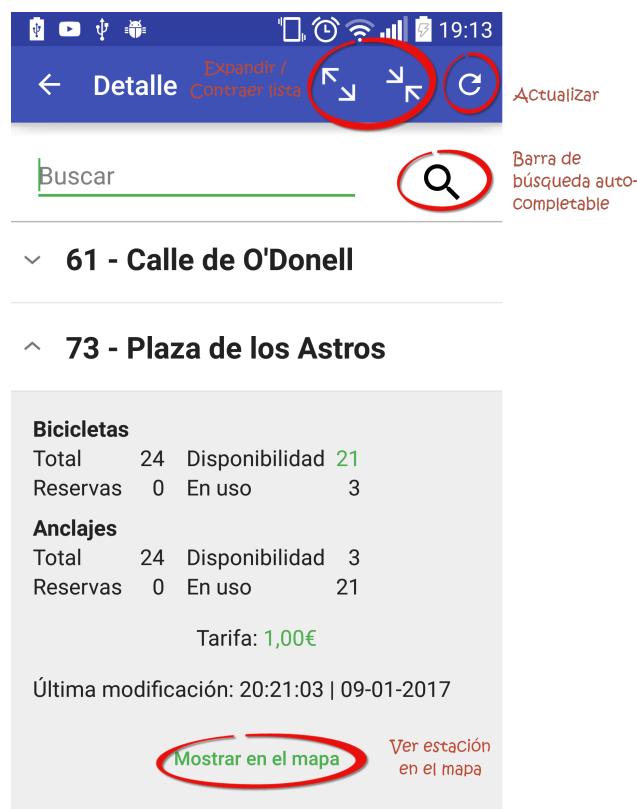


Figura C.8: Listado detallado del conjunto de estaciones

El gráfico permite conocer el estado global del parque de estaciones, tanto para las bicicletas como para los anclajes, mostrando el porcentaje sobre el total de recursos disponibles, ocupados o reservados.

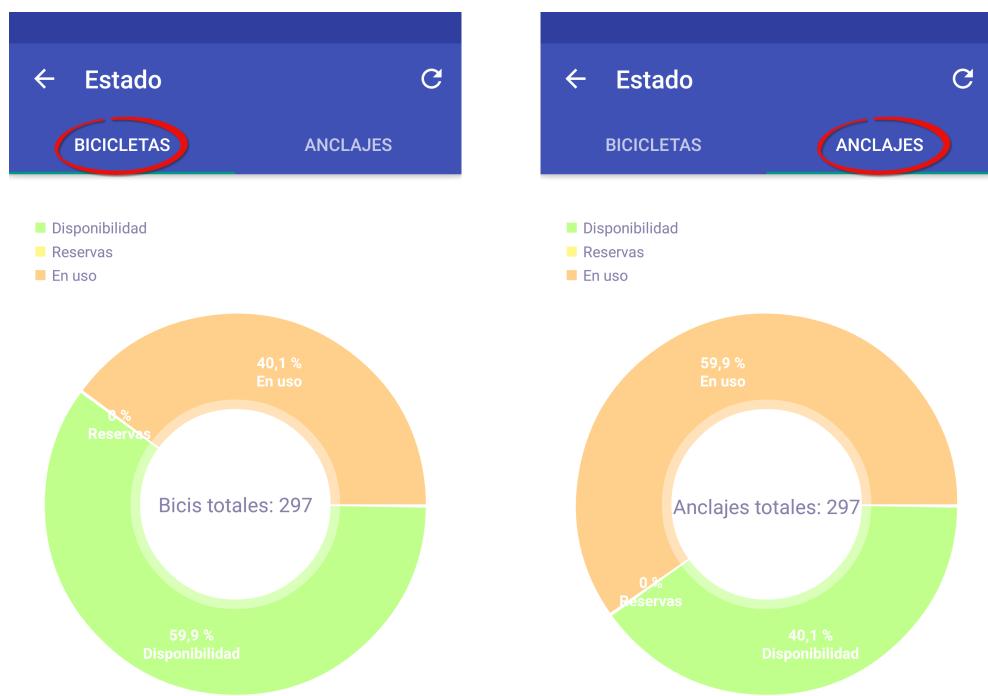


Figura C.9: Estado global del parque de estaciones

Finalmente, el menú desplegable incluye las siguientes opciones adicionales:

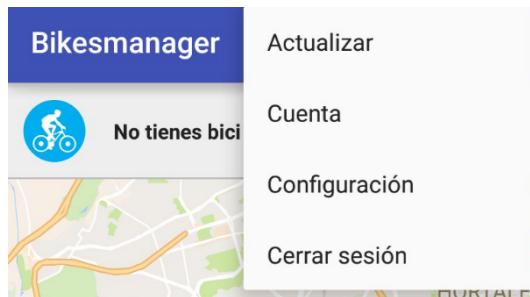


Figura C.10: Menú desplegable y opciones adicionales

- **Actualizar.** Actualiza el estado global del parque de estaciones. Si bien la aplicación se actualiza automáticamente con cada operación, esta opción habilita una actualización explícita. Señalar que las operaciones de actualización sobre las pantallas del listado de estaciones y el gráfico son análogas a esta.

- **Cuenta.** Acceso a la cuenta de usuario. Supone una opción adicional al acceso directo mostrado en la figura C.5.
- **Configuración.** Acceso a diferentes configuraciones del sistema, más adelante en este manual se comenta esta opción en mayor detalle.
- **Cerrar sesión.** Mediante una confirmación adicional por parte del usuario, se cierra la sesión y se vuelve a la pantalla de inicio de sesión.

Operar con bicicletas y anclajes

Mediante la selección de una estación se muestra su estado en ese momento y se habilitan las diferentes opciones sobre la misma: coger una bicicleta, dejar una bicicleta y reservar una bicicleta y/o un anclaje.

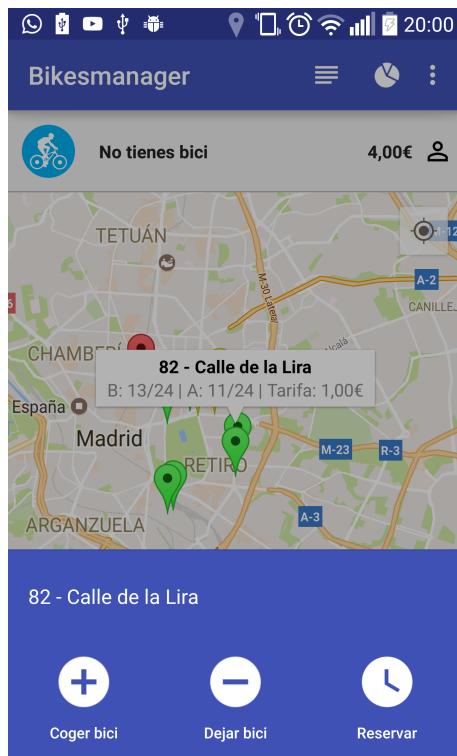


Figura C.11: Opciones a la hora de operar con una estación

De este modo, al coger una bici, se actualizará la barra de estado, además del saldo disponible:



Figura C.12: Actualización de la barra de estado al coger una bici

Barra de estado que volverá a su estado inicial cuando se deje la bicicleta.

Por su parte, al seleccionar la reserva de recursos, se habilitará un diálogo que permitirá seleccionar el elemento que queramos reservar (sólo aparecerán habilitadas para su selección las opciones posibles para el usuario dadas las restricciones consideradas en esta operación, ver la especificación del requisito *RF08 - Reservar* de la sección 2.1 para más detalles). Una vez completada la reserva, la estación afectada quedará marcada en color azul, para una mejor identificación.

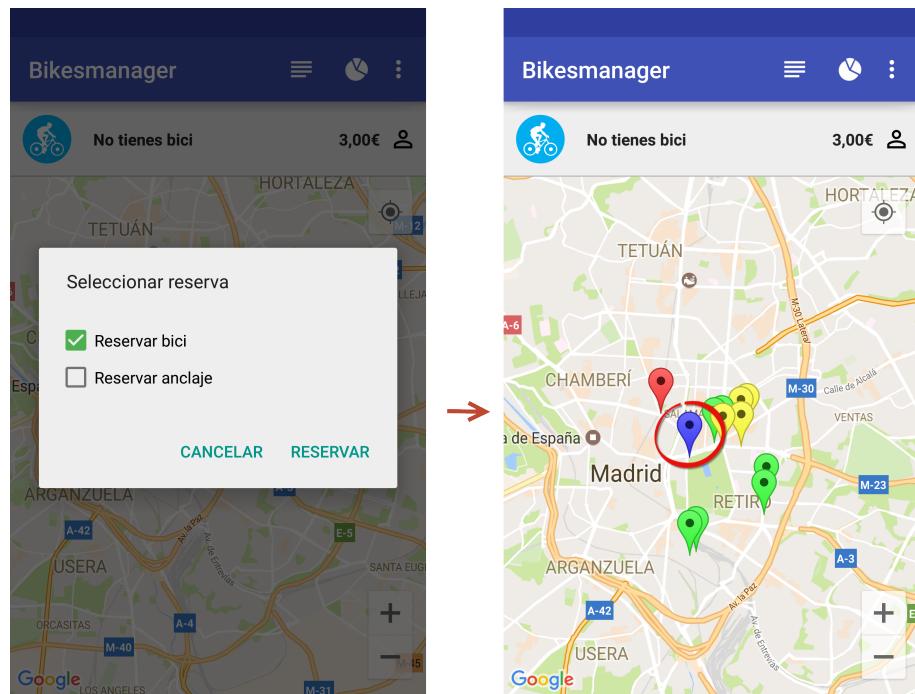


Figura C.13: Reserva de bicicleta y anclajes

Cuenta de usuario

Una vez se haya accedido a la cuenta de usuario mediante alguna de las opciones mencionada, se habilitan las siguientes opciones:

- **Consulta y cancelación de reservas.** La cancelación requiere de una confirmación adicional por parte del usuario.
- **Consulta e ingreso de saldo.**
- **Consulta, edición y borrado del perfil de usuario.** La edición de perfil supone una actividad similar al registro de usuario, mientras que el borrado de la cuenta requiere de una confirmación adicional por parte del usuario.

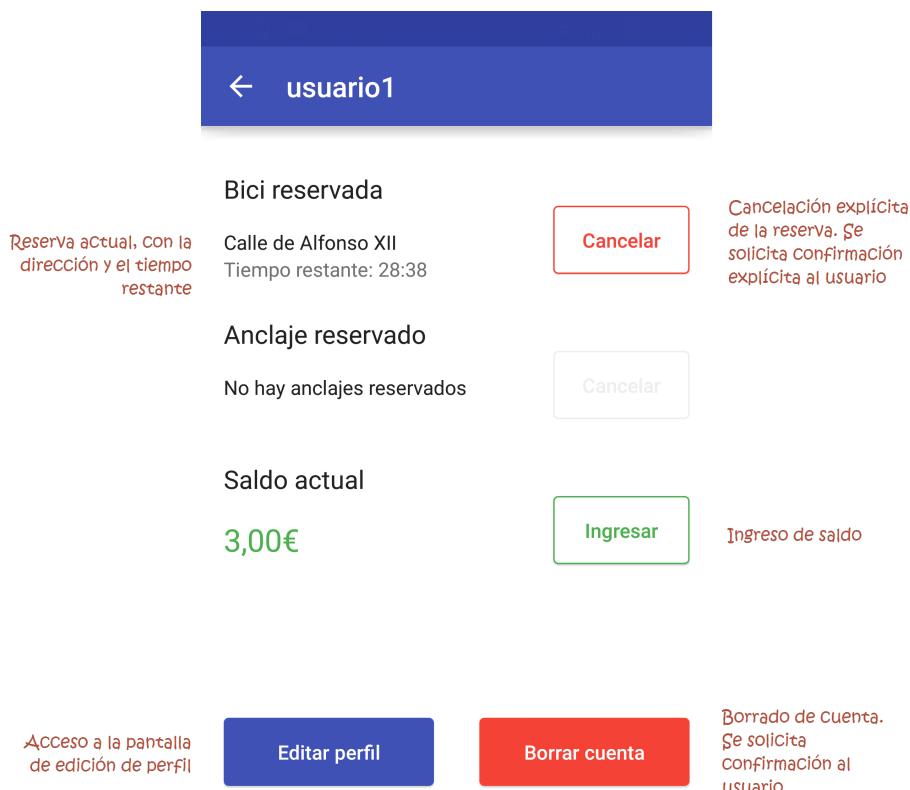


Figura C.14: Cuenta de usuario

Configuración

Mediante el acceso a esta pantalla a través del menú de opciones adicionales comentado antes, se habilitan las siguientes opciones:

- **Cambiar los datos de conexión con el servidor.** Análogo al procedimiento comentado al inicio de este manual.

- **Habilitar el modo “superusuario”.** Mediante la activación de este modo no se comprueban restricciones de usuario a la hora de coger o dejar bicis (se siguen comprobando las restricciones habituales para la reserva de recursos y las relativas a las estaciones, es decir, si una estación no tiene bicis, no se pueden coger). El objetivo de esta opción es el desarrollo de pruebas sobre el software de un modo más ágil.

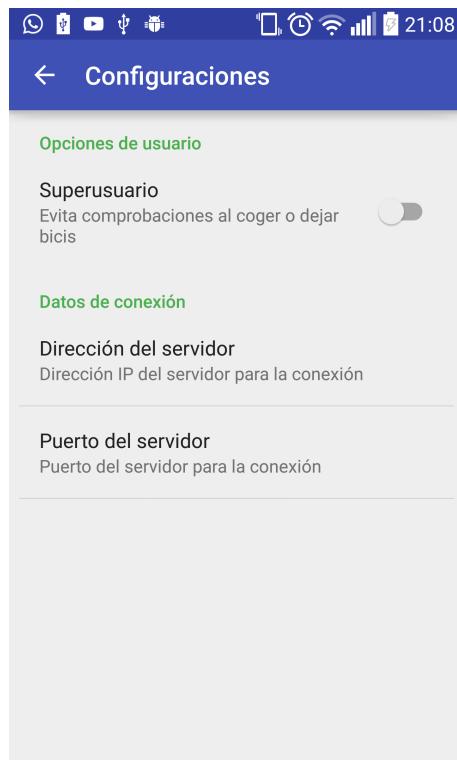


Figura C.15: Configuraciones posibles

En el caso de activar el modo “superusuario”, la barra de estado informará de ello:

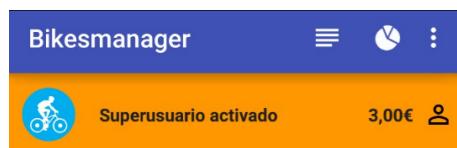


Figura C.16: “Superusuario” activado

FAQs

Se recoge un conjunto de incidencias comunes en el uso básico de la aplicación con su posible resolución, partiendo del supuesto que los servidores de aplicación y base de datos están correctamente configurados y conectados entre sí.

- *Al registrar un usuario, la aplicación no me deja porque el nombre de usuario o dirección de correo no están disponibles.* Hay otro usuario registrado con ese nombre de usuario o dirección de correo, debes introducir uno diferente.
- *Al iniciar sesión, la aplicación no me lo permite por no encontrar el usuario o por contraseña incorrecta.* Asegúrate de que el nombre de usuario y contraseña introducidos para el inicio de sesión coinciden con los registrados.
- *A pesar de estar correctamente conectado, no puedo coger una bici.* Es posible que estés incurriendo en alguna de las restricciones impuestas para esta operación, ver la especificación del requisito *RF06 - Coger bicicleta* de la sección 2.1 para más detalles.
- *A pesar de estar correctamente conectado, no puedo dejar una bici.* Es posible que estés incurriendo en alguna de las restricciones impuestas para esta operación, ver la especificación del requisito *RF07 - Dejar bicicleta* de la sección 2.1 para más detalles.
- *A pesar de estar correctamente conectado, no puedo reservar una bicicleta y/o anclaje.* Es posible que estés incurriendo en alguna de las restricciones impuestas para esta operación, ver la especificación del requisito *RF08 - Reservar* de la sección 2.1 para más detalles.
- *Tenía una reserva que no he cancelado y ha desaparecido a pesar de que no he hecho uso de ella.* Posiblemente habrán pasado más de 30 minutos desde la reserva. Este tiempo es el máximo establecido para hacer uso de la reserva, una vez superado se cancela automáticamente.
- *No puedo borrar mi perfil.* Es posible que tengas una bicicleta cogida, debes dejarla en una estación antes de borrar el perfil.
- *He borrado mi cuenta, ¿puedo recuperarla?* No, el borrado de cuenta, una vez confirmado por el usuario, es definitivo, con lo que el acceso a la aplicación sólo se puede realizar registrando uno nuevo.

Apéndice D

Manual de instalación

Se aporta un manual de instalación del sistema, de modo que se pueda desplegar en otros equipos con facilidad. Cabe señalar que las instrucciones están redactadas siguiendo un entorno Windows, el despliegue en entornos iOS o Unix pueden acarrear, por tanto, ciertas diferencias.

Base de datos

1. Descargar e instalar MySQL Installer desde <http://dev.mysql.com/downloads/windows/installer/>.
2. Durante la instalación, es posible que la aplicación trate de instalar o actualizar otros productos MySQL. Se recomienda ignorar estos procesos durante la instalación y dejarlos una vez MySQL Installer esté listo para ser ejecutado en nuestro equipo puesto que son susceptibles de fallar.
3. Al ejecutar MySQL Installer, aparecerán los productos MySQL actualizables o instalables, se deben añadir:
 - a) El servidor MySQL Server.
 - b) El conector Connector/J.
 - c) El entorno MySQL Workbench.

En caso de que se quiera reiniciar algún componente, desde esta herramienta se puede eliminar para volverlo a instalar a continuación. En la siguiente imagen falta por añadir el servidor, mientras que el conector y el Workbench están al día:

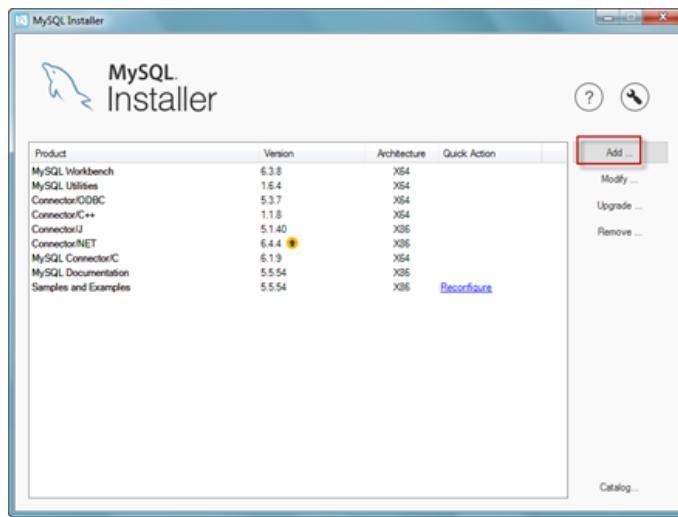


Figura D.1: Pantalla principal de MySQL Installer

4. Al instalar el servidor, se pedirá la confirmación de una serie de parámetros de instalación. Se puede mantener la configuración que aparece por defecto:

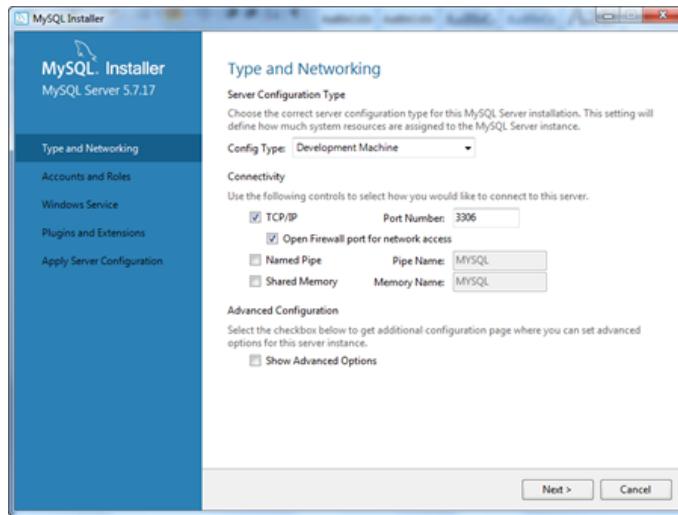


Figura D.2: Instalación de MySQL Server

5. A continuación se pedirá la contraseña para el usuario *root*, no es necesario añadir usuarios. **Importante recordar esta contraseña**, puesto que se necesitará más adelante para configurar el servidor.
6. El resto de pasos se pueden pasar sin realizar modificaciones, hasta el paso

final donde se ha de ejecutar una secuencia de acciones que da por finalizada la instalación:

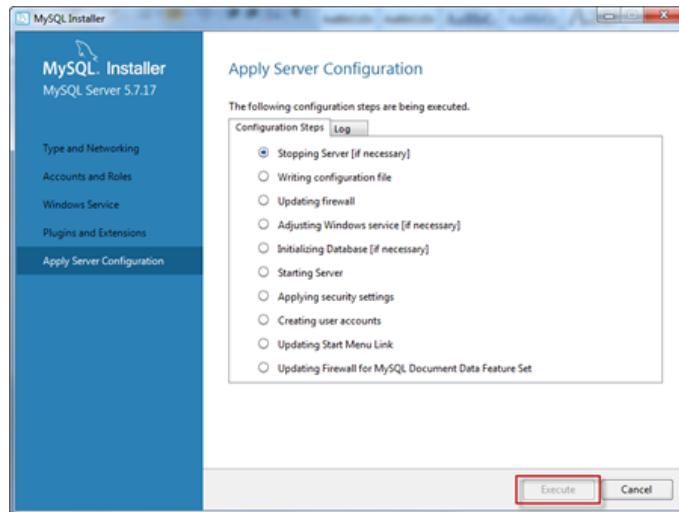


Figura D.3: Instalación de MySQL Server

7. Abrir MySQL Workbench y pulsar en la pestaña superior derecha de la conexión que aparece configurada por defecto (localhost:3306) para testearla. En caso de que pida **contraseña**, introducir la misma que la seleccionada para el servidor MySQL Server anterior.

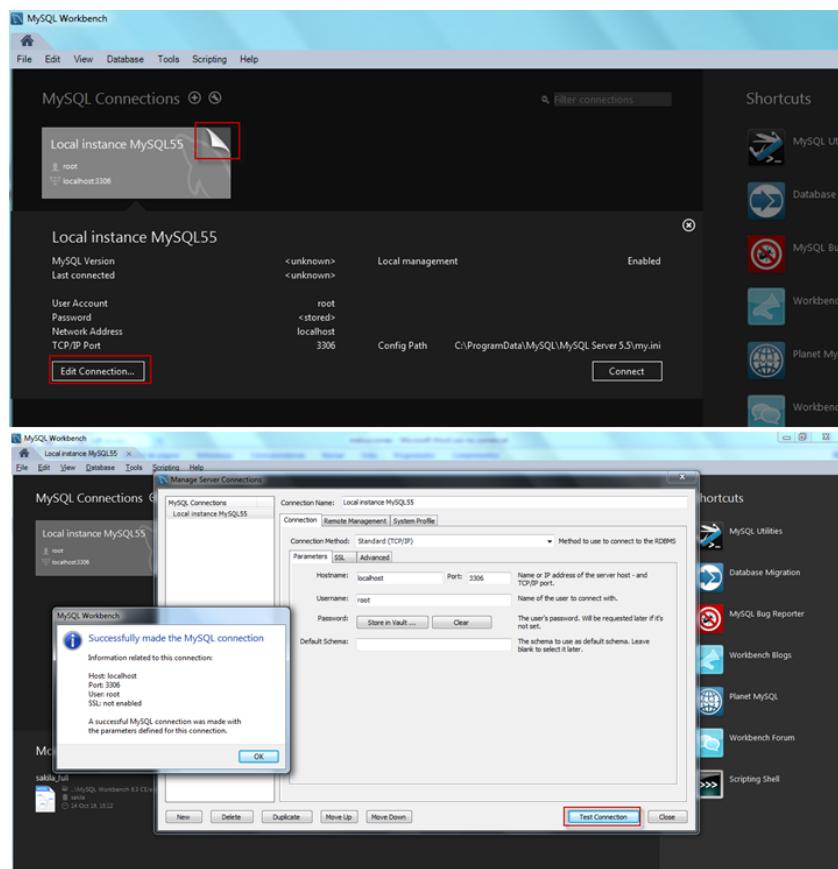
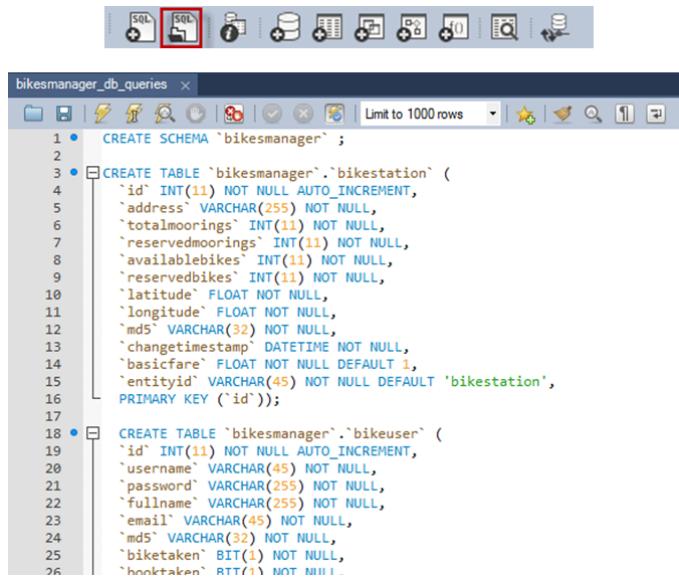


Figura D.4: Configuración de la conexión en MySQL Workbench

8. Doble click en la conexión para acceder.
9. Cargar y ejecutar las consultas SQL del fichero “bikesmanager_db_queries.sql”:



```

bikesmanager_db_queries x
CREATE SCHEMA `bikesmanager` ;
CREATE TABLE `bikesmanager`.`bikestation` (
    `id` INT(11) NOT NULL AUTO_INCREMENT,
    `address` VARCHAR(255) NOT NULL,
    `totalmoorings` INT(11) NOT NULL,
    `reservedmoorings` INT(11) NOT NULL,
    `availablebikes` INT(11) NOT NULL,
    `reservedbikes` INT(11) NOT NULL,
    `latitude` FLOAT NOT NULL,
    `longitude` FLOAT NOT NULL,
    `md5` VARCHAR(32) NOT NULL,
    `chagetimestamp` DATETIME NOT NULL,
    `basicfare` FLOAT NOT NULL DEFAULT 1,
    `entityid` VARCHAR(45) NOT NULL DEFAULT 'bikestation',
    PRIMARY KEY (`id`));
CREATE TABLE `bikesmanager`.`bikeuser` (
    `id` INT(11) NOT NULL AUTO_INCREMENT,
    `username` VARCHAR(45) NOT NULL,
    `password` VARCHAR(255) NOT NULL,
    `fullname` VARCHAR(255) NOT NULL,
    `email` VARCHAR(45) NOT NULL,
    `md5` VARCHAR(32) NOT NULL,
    `biketaken` BIT(1) NOT NULL,
    `hanktaken` BTY(1) NOT NULL);

```

Figura D.5: Ejecución de consultas SQL

En caso de que no se actualice el menú izquierdo referente a los esquemas: click derecho sobre el menú y “Refresh All”.

- Para cargar los datos de bicis, utilizar el fichero “bikesmanager_db_bicis.csv” (los usuarios y las reservas se crean durante la ejecución) pulsando sobre el botón de ejecución de consultas sobre la tabla “bikestation” (aparece al pasar el ratón por encima). **Eliminar la primera fila que se carga con las cabeceras** (botón derecho y “Delete rows”) y, finalmente, pulsar en “Apply”.

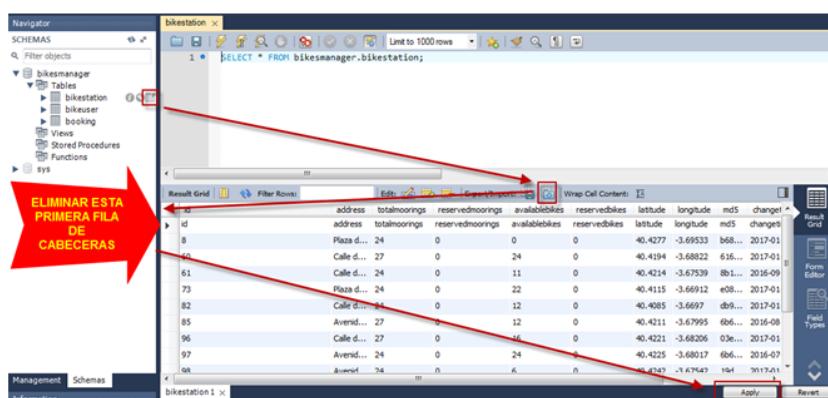


Figura D.6: Carga de datos en MySQL Workbench

Servidor

1. Descargar NetBeans IDE desde <https://netbeans.org/downloads/>. Se recomienda descargar la versión completa, de 221MB.
2. Aceptar la instalación por defecto, que ya incluye GlassFish.
3. Copiar (sustituyendo el existente) el fichero “org.eclipse.persistence.moxy” en el directorio “modules” de GlassFish. Típicamente este fichero se encuentra en la ruta “... \glassfish-4.1.1\glassfish\modules”. El motivo es que GlassFish cuenta con un fallo no resuelto a fecha de redacción del documento sobre la producción de cadenas JSON que se soluciona con la sustitución anterior.
4. Ahora se debe configurar la conexión con la base de datos en GlassFish (es importante asegurar MySQL Server está corriendo, se recomienda acceder primero con MySQL Workbench a la base para asegurarse de ello). La ruta en NetBeans sería: Services - Databases - Drivers. Click derecho sobre “MySQL (Connector/J Driver)” y “Connect Using...”:

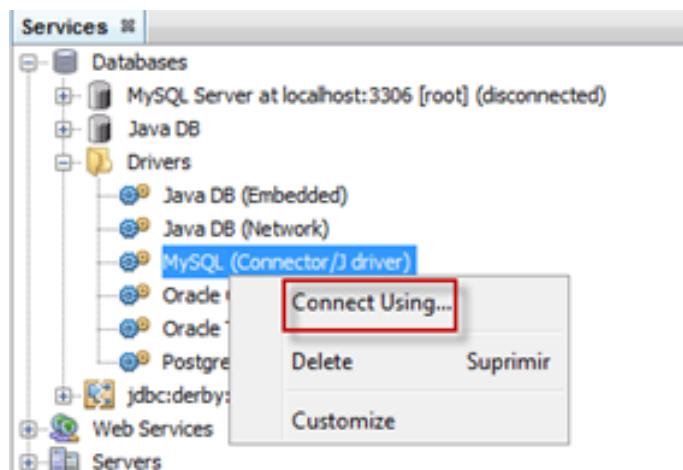


Figura D.7: Conexión del Connector/J

5. Se introduce el nombre de la Base de datos (“bikesmanager”) y **la contraseña elegida al configurar MySQL Server para el usuario root** (y que también se utilizó para acceder a MySQL Workbench). Se recomienda dejarla recordada. El resto de parámetros se mantienen por defecto (localhost:3306). Pulsar en “Test Connection” para confirmar la conexión.

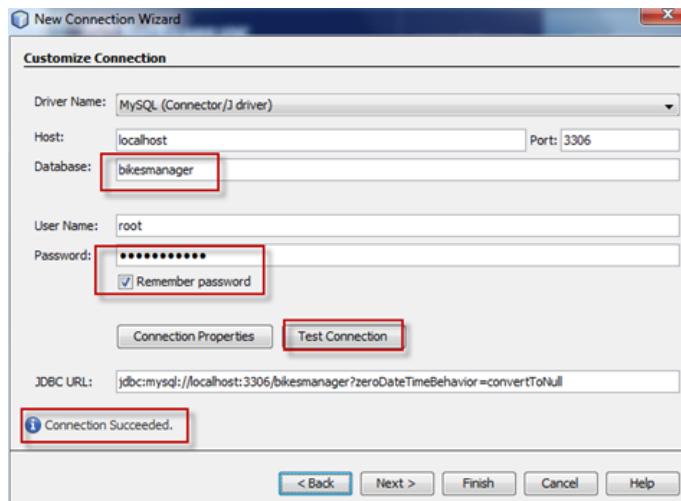


Figura D.8: Configuración de la conexión

6. Pulsar en “Next” hasta llegar a la última ventana, donde se pulsa “Finish”. Ahora desde el IDE NetBeans se tiene acceso a gestionar las tablas (además de con MySQL Workbench).

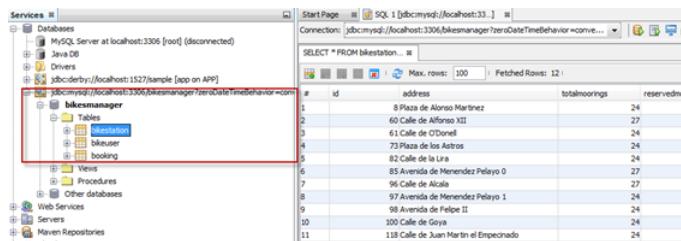


Figura D.9: Vista de la base de datos desde NetBeans

7. Finalmente, se debe realizar la conexión a MySQL Server, para ello: botón derecho sobre “MySQL Server at...” y “Properties”. Introducir la **contraseña del servidor MySQL Server** y dejarla recordada:

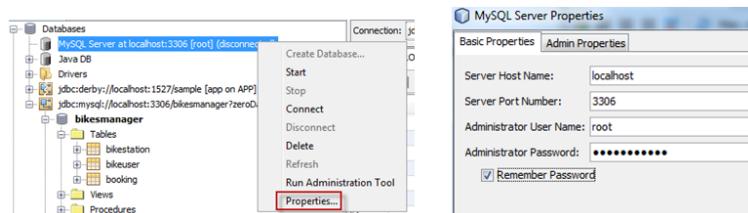


Figura D.10: Conexión base de datos

8. Aceptar y, de nuevo, botón derecho sobre la base y “Connect”. Se podrá ver el esquema de base de datos una vez hecho esto:

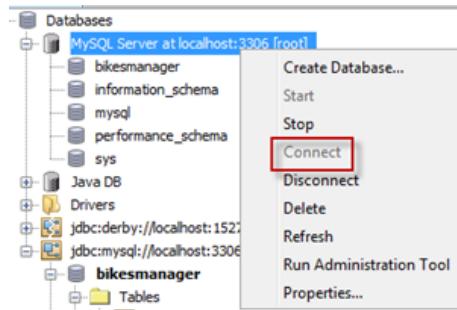


Figura D.11: Conexión base de datos

9. Cargar el proyecto mediante la ruta siguiente: File – Import Project – From ZIP...:

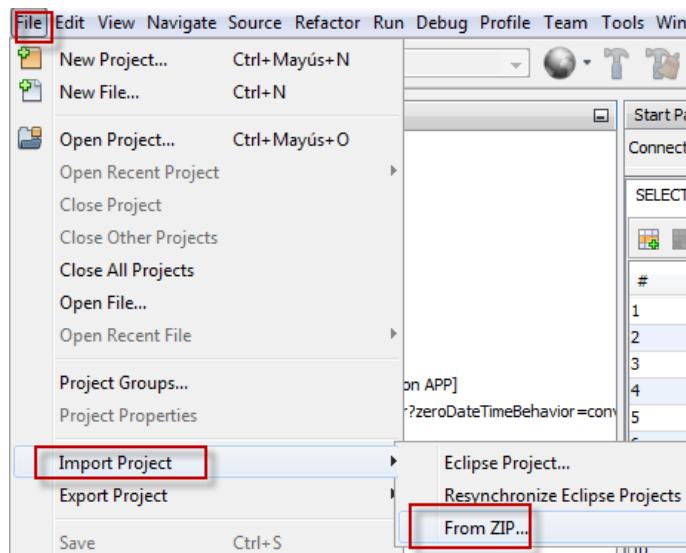


Figura D.12: Carga del proyecto

10. Seleccionar “bikesmanager_server.zip” e importar.
11. Dentro del proyecto, abrir el fichero “glassfish-resoruces.xml” en la carpeta “Configuration Files” y poner la **contraseña del servidor MySQL Server** en el campo correspondiente:



```

<resources>
    <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread="true"
        <property name="serverName" value="localhost"/>
        <property name="portNumber" value="3306"/>
        <property name="databaseName" value="mysql"/>
        <property name="User" value="root"/>
        <property name="Password" value="CONTRASEÑA DEL SERVIDOR MYSQL SERVER"/>
        <property name="URL" value="jdbc:mysql://localhost:3306/mysql?zeroDateTimeBehavior=convertToNull"/>
        <property name="driverClass" value="com.mysql.jdbc.Driver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="java:app/BikesManagerDB" object-type="Container">
        <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread="true"
            <property name="serverName" value="localhost"/>
            <property name="portNumber" value="3306"/>
            <property name="databaseName" value="bikesmanager"/>
            <property name="User" value="root"/>
            <property name="Password" value="CONTRASEÑA DEL SERVIDOR MYSQL SERVER"/>
            <property name="URL" value="jdbc:mysql://localhost:3306/bikesmanager?zeroDateTimeBehavior=convertToNull"/>
        </jdbc-connection-pool>
    </jdbc-resource>
</resources>

```

Figura D.13: Configuración de claves en “glassfish-resoruces.xml”

El resto de parámetros se pueden dejar tal y como aparecen puesto que muestran los valores por defecto establecidos anteriormente.

12. Compilar el proyecto.
13. Si aparece un ícono de advertencia sobre el proyecto, es posible que se haya quedado una conexión sin establecer. Se debe pulsar botón derecho sobre el mismo y, a continuación, “Resolve data source problems” (en la parte inferior). En el menú que aparece, pulsar “Add connection”. Esto no supone un problema, pero mejor eliminar las advertencias.
14. Encender GlassFish. Adicionalmente, pulsando botón derecho sobre el servidor podemos acceder a sus “Properties” para cambiar la IP sobre la que escucha. Por defecto GlassFish escucha en localhost:8080:

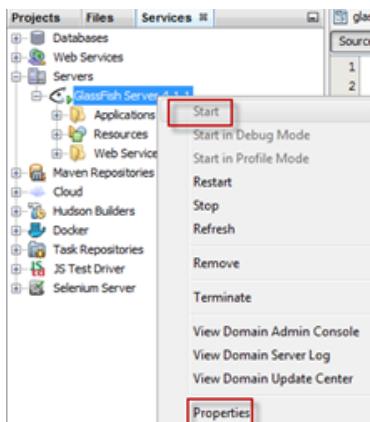


Figura D.14: Encendiendo GlassFish

15. Desplegar el proyecto. Ahora desde el navegador se puede probar la conexión introduciendo, por ejemplo, la siguiente URL: <http://localhost>:

8080/BikesManager/rest/entities.bikestation que ha de devolver el conjunto de estaciones en una cadena JSON.

Cliente

1. Guardar el archivo “bikesmanager_installer.apk” en el dispositivo mediante, por ejemplo, una conexión USB.
2. Abrir el archivo en la localización elegida para proceder a su instalación. (NOTA: si se tiene activa alguna app de atenuación de pantalla es posible que el botón de instalación no esté accesible, deshabilitar la app para poder proceder).

FAQs

Se recoge un conjunto de incidencias comunes identificadas en el proceso de instalación.

- *Tengo la aplicación descargada en mi dispositivo, pero el botón “Instalar” no reacciona cuando lo pulso.* Es posible que tengas instalada y activa una aplicación de atenuación de pantalla, ponla en pausa para poder pulsar el botón.
- *Si bien antes no había problema, ahora no soy capaz de conectar el servidor de aplicación con la base de datos, no siendo posible desplegar el proyecto sobre GlassFish.* Es posible que se haya realizado un cambio de configuración o seguridad sobre la base de datos o sobre los datos de conexión del servidor sobre ella. Este error es típico si se ha realizado un cambio de contraseña para el acceso a la base de datos. Asegura los siguientes puntos:
 - Que el servidor de base de datos está ejecutándose.
 - Asegurar que los datos de conexión con la base de datos configurados (IP (localhost si están el mismo equipo), puerto (3306 típicamente para MySQL), contraseña (la utilizada para el acceso a la base), etc.) están correctamente establecidos, tanto en GlassFish como en el fichero “glassfish-resources.xml” del proyecto desplegado sobre el servidor.Es importante, por tanto, notar que ante un cambio en la contraseña para el acceso a la base de datos, se ha de modificar también este dato tanto en la conexión configurada en GlassFish con la base como en el fichero “glassfish-resources.xml” del proyecto (en el manual de instalación del servidor se pueden consultar los elementos que hacen uso de esta clave y que, por tanto, se han de actualizar).

- *Una vez introducidos los datos de conexión, no puedo operar con la aplicación, que me informa de que no se ha podido establecer la conexión con el servidor.*

Es posible que no se esté efectuando correctamente la conexión entre la aplicación y el servidor, asegura los siguientes puntos:

- Que la IP y el puerto son correctos, es decir, se corresponden con los utilizados por el servidor de aplicación GlassFish. Si ejecuta en localhost, la IP a utilizar será la asignada a tu equipo en la red a la que esté conectado.
- Que el servidor de aplicación está ejecutándose.
- Que el firewall de tu equipo no está bloqueando la conexión. En tal caso, añade una excepción, tanto de entrada como de salida, para el ejecutable de Java (plataforma sobre la que ejecuta GlassFish) en el puerto en el que esté escuchando el servidor (típicamente el 8080). Otra opción es desconectar el firewall durante el ejecución.

Bibliografía

- [AnDev] Portal de Android Developers en <https://developer.android.com>.
- [Arl02] Arlow, J. y I. Neustadt, *UML and the Unified Process*, Addison-Wesley, 2002.
- [Bas03] Bass, L., P. Clements y R. Kazman, *Software Architecture in Practice*, 2a. ed., Addison-Wesley, 2003.
- [Bei90] Beizer, B., *Software Testing Techniques*, 2a. ed., Van Nostrand-Reinhold, 1990.
- [Coa91] Coad, P. y E. Yourdon, *Object-Oriented Analysis*, 2a. ed., Prentice Hall, 1991.
- [Coo00] Cooper, J. W., *Java Design Patterns: A Tutorial*, Addison-Wesley Professional, 2000.
- [DocOr] Portal de documentación técnica Oracle sobre Java en <http://docs.oracle.com/javase>.
- [EckPa] Eckel, B. *Thinking in Patterns*.
- [Gar87] Garvin D., *Competing on the Eight Dimensions of Quality*, Harvard Business Review, noviembre 1987.
- [GF13] *GlassFish Server Open Source Edition - Quick Start Guide*, 4.0, 2013.
- [Git] Portal de Git en <https://git-scm.com/>
- [GitHb] Portal de GitHub en <https://github.com/>
- [IE830] Institute of Electrical and Electronics Engineers (IEEE), *IEEE Std. 830-1998: Especificaciones de los Requisitos del Software*.

- [ISO25] International Organization for Standardization (ISO), *ISO/IEC 25010:2011 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.*
- [JSON] “Introducción a JSON” en <http://www.json.org/json-es.html>.
- [Kan99] Kaner, C., J. Falk y H. Q. Nguyen, *Testing Computer Software*, 2a. Rev. ed., John Wiley & Sons, 1999.
- [Man97] Mandel, T., *The Elements of User Interface Design*, John Wiley & Sons, 1997.
- [McC77] McCall, J., P. Richards y G. Walters, *Factors in Software Quality, Volume I. Concepts and Definitions of Software Quality*, 1977.
- [Mye11] Myers, G., C. Sandler, y T. Badgett, *The Art of Software Testing*, 3a. ed., John Wiley & Sons, 2011.
- [NetRF] “Getting Started with RESTful Web Services” en <https://netbeans.org/kb/docs/websvc/rest.html>.
- [Oet15] Oetiker T. et al, *The Not So Short Introduction to LATEX 2ε*, 5.05, 2015.
- [OraDB] “Java SE Technologies - Database” en <http://www.oracle.com/technetwork/java/javase/jdbc/index.html>.
- [OraEE] “Java EE at a Glance” en <http://www.oracle.com/technetwork/java/javaee/overview/index.html>.
- [Pre10] Pressman, R. S., *Ingeniería del software: un enfoque práctico*, 7a. ed., McGrawHill, 2010.
- [Sha96] Shaw, M. y D. Garlan, *Software Architecture*, Prentice Hall, 1996.
- [WREST] “Representational state transfer” en https://en.wikipedia.org/wiki/Representational_state_transfer.
- [WSOAP] “SOAP” en <https://en.wikipedia.org/wiki/SOAP>.
- [WWSer] “Web Service” en https://en.wikipedia.org/wiki/Web_service.