UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

SmartBeds

Aplicación de técnicas de minería de datos para la detección de crisis epilépticas

Anexos

Presentado por José Luis Garrido Labrador
en Universidad de Burgos – 13 de junio
de 2019
Tutores: Dr. Álvar Arnaiz González
y Dr. José Francisco Díez Pastor

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apéndice se expondrán los distintos *sprints* que se han realizado y un estudio de la viabilidad del proyecto.

A.2. Planificación temporal

El proyecto se desarrolló siguiendo una metodología ágil basada en *Scrum*. Se dividió el progreso en *Sprints*, cada uno con una serie de tareas y su estimación en esfuerzo. Esta estimación o peso, se ha evaluado según la dificultad que se preveía tener, no como una medida horaria, esto es debido que algunas tareas simples tardan más en desarrollarse por la necesidad de la ejecución automática mientras que otras que requirieron mucho más trabajo se ejecutaron mucho antes. Algunas tareas además se consideran épicas, estas son resúmenes de un proceso completo que abarca varias tareas y duran generalmente más de un *sprint*.

A continuación se mostrarán las listas de las tareas realizadas con los enlaces a las *issues* del repositorio *GitHub*. Los *Sprints* fueron:

***Sprint* 1 - 10/11/18 hasta 20/11/18**

En este primer *Sprint* solamente hubo dos tareas:

1. Hacer exploración bibliográfica
2. Configurar repositorio de Git

Sprint 2 - 21/11/18 hasta 05/12/18

Se desarrollaron 4 tareas, todas de investigación, por lo que no hay *commits* asociados, para esto las tareas fueron:

3. Lectura de «Automated Epileptic Seizure Detection Methods: A Review Study»
4. Exploración Bibliográfica - Orientado a caídas
5. Lectura del primer tema de "Minería de Datos"
6. Configurar VPN en Archlinux

Sprint 3 - 06/12/18 hasta 21/12/18

En este *sprint* se comenzó la documentación migrando las plantillas al repositorio y continuó la investigación a un área más técnica, las tareas fueron:

7. Iniciar documentación
8. Tabla de extracción de características
9. Búsqueda de librerías con funciones sofisticadas

Sprint 4 - 22/12/18 hasta 08/01/2019

Este *sprint* fue del aprendizaje de técnicas de minería de datos aplicada. Las tareas realizadas fueron:

10. Instalación de entorno Python
11. Graficar datos mediante PCA
12. Aprender el uso de librerías

Sprint 5 - 09/01/2019 hasta 17/01/2019

En este *sprint* el objetivo fue probar distintas lineas de investigación para ver las características intrínsecas a los datos.

13. Configurar acceso a gamma

14. Leer apuntes de Minería de Datos
15. Filtrado y suavizado de datos
16. Probar otras formas de proyección

Sprint 6 - 18/01/2019 hasta 24/01/2019

Tras descartar algunas proyecciones se exploraron las más prometedores y se probaron nuevos filtros y detección de anomalías. Las tareas fueron por consiguiente:

17. Mejor preprocesamiento
18. Dibujado alrededor de las crisis
19. Probar formas de filtrado
20. Estudiar puntos clave de las proyecciones
21. Probar detección de anomalías por one-class

Sprint 7 - 25/01/2019 hasta 07/02/2019

Este *sprint* tuvo una duración mayor debido a que durante el periodo señalado se realizó un curso en la universidad donde se estudiaron las series temporales. Por este motivo, el servidor donde se han estado realizando las ejecuciones no estaba disponible retrasando la ejecución de los experimentos.

Las tareas se centraron en comprobar las proyecciones más interesantes con datos estadísticos además de profundizar en *One-Class* o detección de anomalías. Las tareas realizadas fueron:

22. Proyección LTSA con valores estadísticos [6]
23. Documentar Sprints del 1 al 6
24. Documentar investigación con Alicia Olivares
25. One Class - Mejorar la investigación de este apartado
26. Transformers
27. Trasladar códigos a transformers

Sprint 8 - 8/02/2019 hasta 14/2/2019

Este *sprint* fue dedicado a seguir desarrollando el estudio de *One Class* cuyos resultados se pueden ver en el apéndice *Cuaderno de Investigación*. Las tareas fueron:

28. Particionar los datos
29. Estudiar one-class[1]
30. One-Class con crisis
31. One-Class sin crisis
32. Preprocesado básico para *One Class*
    a) Bruto
    b) Filtrado *Butter* de 3 y 0.05
    c) Filtrado *SavGol* de tamaño 15
    d) Concatenación de estadísticos en ventana 25 sobre bruto
    e) Concatenación de estadísticos con ventana 25 sobre *Butter* de 3 y 0.05
    f) Concatenación de estadísticos con ventana 25 sobre *SavGol* de tamaño 15
33. Testear clasificadores con otras crisis

Las tareas de la 30 a la 33 se incluyeron en la tarea épica 29 aunque esta tuvo tareas del siguiente *sprint*

Sprint 9 - 15/2/2019 a 21/2/2019

Este *sprint* trató de completar la linea de investigación de *One Class* abierta en la tarea épica 29 y documentar los resultados de *sprints* anteriores.

34. Documentar PCA y Proyecciones de 2-Variedad
35. Entrenamiento *One Class* con dos crisis
36. Testeo *One Class* con tercera crisis

[1] Tarea épica, siendo estas las tareas que engloban varias tareas más simples

37. Calcular área bajo la curva con entrenamiento de una clase
38. Visualización de constantes vitales

Son las tareas de la 35 a la 37 las que forman parte de la tarea épica 29.

Sprint 10 - 22/3/2019 a 28/3/2019

Este *sprint* estuvo centrado en la lectura y aprendizaje de nuevos métodos

39. Investigar sobre desequilibrados
40. Aprender Weka
41. Documentar *One Class*

Sprint 11 - 01/03/2019 a 21/03/2019

Este *sprint* tuvo una duración mayor debido a la diversidad de experimentos y la presencia de varios días no lectivos. Se centró en el lanzamiento de experimentos con *ensembles*.

42. Creación de test de transformers
43. Filtrado SMOTE
44. Pruebas con ensembles ya implementados
45. Prueba de ensembles para desequilibrados

Sprint 12 - 22/03/2019 - 28/03/2019

En esta ocasión comenzamos a reducir el esfuerzo sobre la investigación sobre un incremento en el diseño de la aplicación.

46. Exploración de herramientas
47. Documentar resultados anteriores
48. Diseño del servidor
49. Ejecutar experimentos Weka restantes

Sprint 13 - 29/03/2019 - 08/04/2019

Este *sprint* se centró en la definición de requisitos y de realizar experimentos con los resultados de Alicia Olivares. Sin embargo, estos experimentos no pudieron ser realizados al descubrir un fallo en la investigación previa, por lo que no se realizaron quedando pendientes ante nuevos datos. Por tanto, las tareas que realmente fueron realizadas fueron:

50. Simulador de la cama
52. Diseño de casos de uso
53. Diseño de los datos para la API
54. Diseño de la base de datos
57. Diseño de la API
58. Requisitos funcionales

Spint 14 - 09/04/2019 - 11/04/2019

Este *sprint* se concluyeron la creación de los experimentos que quedaron pendientes en el *sprint* anterior como seguir desarrollando componentes de la aplicación.

Sin embargo, muchas partes de este *sprint* fueron quedando en *icebox* tras algunas complicaciones en los experimentos y la nueva linea de investigación con *PRC* que se puede ver en el *Cuaderno de investigación* adjuntado.

51. Experimentos Weka con los datos de series temporales
55. Implementación de la base de datos Creación de experimentos
58. Lanzamiento de experimentos
59. Documentación de resultados
60. Experimentos Desequilibrador[2]
61. Ultimar detalles del cuaderno de trabajo
62. Arreglar documentación de casos de uso

[2]Tarea épica

63. Completar diseño de bases de datos

64. Trabajar en formato de latex

65. Crear interfaz de la api

66. Creación y documentación de prototipos de interfaz

Spint 15 - 12/04/2019 - 2/05/2019

En este *sprint* se comenzó la implementación de la API así como la realización de test automáticos de la misma. También se comenzó a preparar el servidor para que fuese accesible desde Internet.

68. Programación de la API

69. Test de API

70. Acceso web a la API

71. Documentar Sprints anteriores

72. Configurar servidor

73. Entorno Flask

Spint 16 - 3/05/2019 - 09/05/2019

Este *sprint* consistió en continuar el desarrollo del servidor.

74. Hilo de procesamiento

75. Distribución de paquetes por SocketIO

76. Uso del clasificador de Alicia Olivares

77. Actualización de Chart.js es muy lenta

78. Lanzar proyecto en el servidor

Spint 17 - 10/05/2019 - 16/05/2019

Este *sprint* se centró en el desarrollo de la vista así como reparar algunos *bugs* que fueron apareciendo. También se intento utilizar *Heroku* pero al conseguir eliminar las limitaciones del servidor se desechó la idea.

80. Conexiones SQL por request
81. Hacer todas las ventanas
82. Crear cama simulada de los datos alrededor de las crisis
83. Escribir memoria - Herramientas
84. Error en conexiones por request: MySQL Connection Not Available

Spint 18 - 17/05/2019 - 30/05/2019

Este *sprint*, que duró dos semanas, se centró en hacer algunas mejoras sobre la aplicación así como documentar algunas partes de la memoria.

85. Guardar probabilidades del imblearn
86. Censurar opciones de administración de camas
87. Sistema de alertas y control en frontend
88. Simulación falseada de las camas
89. Clasificador aleatorio
90. Escribir el abstract de la memoria
91. Documentar proceso de instalación
92. Concluir ventanas restantes

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

El primer paso de todo desarrollo de software es definir bien los requisitos de la aplicación así como los detalles generales que se necesitan de la aplicación. En este apartado se desarrollaran los requisitos y los casos de uso.

B.2. Objetivos generales

La aplicación web tiene lo siguientes objetivos:

- Monitorizar en tiempo real a los pacientes con epilepsia.
- Gestionar los distintos accesos del personal médico a los datos de los pacientes.
- Ofrecer una API REST de la aplicación.
- Crear una interfaz web fácil e intuitiva.

B.3. Catalogo de requisitos

En esta sección se presentan los requisitos funcionales y los no funcionales

Requisitos funcionales

- **RF-1 Confidencialidad del sistema:** solamente usuarios autorizados podrán acceder al sistema.

  - **RF-1.1 Identificación de usuario:** los usuarios se identificarán con un *nickname* y una contraseña
  - **RF-1.2 Rol de administración:** existirá un usuario especial que podrá administrar el sistema completamente sin restricciones.
  - **RF-1.3 Visualización de una cama:** los usuarios validados deben poder observar los datos en tiempo real de las camas disponibles.
  - **RF-1.4 Restricción de acceso:** los usuarios solamente podrán tener acceso a los datos de las camas permitidas.
  - **RF-1.5 Acceso completo al administrador:** el administrador debe poder acceder a todas las camas existentes.

- **RF-2 Gestión de las camas:** el administrador ha de gestionar las camas pudiendo añadir, modificar y borrar.
  - **RF-2.1 Añadir cama:** el administrador ha de poder añadir una nueva cama al sistema.
  - **RF-2.2 Modificar cama:** el administrador ha de poder modificar los datos una cama existente.
  - **RF-2.3 Borrar cama:** el administrador ha de poder borrar una cama del sistema.
  - **RF-2.4 Asignar camas a usuarios:** el administrador se encarga de decidir que usuario puede acceder a que cama.

- **RF-3 Gestión de los usuarios:** el administrador ha de gestionar los usuarios pudiendo añadir, modificar y borrar.
  - **RF-3.1 Añadir usuario:** el administrador ha de poder añadir un nuevo usuario al sistema.
  - **RF-3.2 Modificar usuario:** el administrador ha de poder modificar los datos un usuario existente.
  - **RF-3.3 Borrar usuario:** el administrador ha de poder borrar un usuario del sistema.

- **RF-4 Visualización de los datos:** los usuarios han de poder ver de las camas disponibles el estado actual del paciente, sus constantes vitales y las presiones.

Requisitos no funcionales

- **RNF-1 Usabilidad:** la aplicación debe cumplir estándares de usabilidad teniendo una curva de aprendizaje baja y un uso de metáforas adecuado.

- **RNF-2 Disponibilidad:** las camas existentes han de ser siempre accesibles por sus usuarios asociados y dar una información correcta de su estado
- **RNF-3 Confidencialidad:** los datos de las camas, al ser en parte constantes vitales de pacientes, solamente han de ser accesibles por los usuarios autorizados.
- **RNF-4 Escalabilidad:** el sistema debe ser escalable para adaptarse mejor a un incremento de carga del sistema.
- **RNF-5 Seguridad:** los usuarios deben poder identificarse sólidamente con el sistema sin que sus datos o sus credenciales (*tokens*) sean accesibles por terceros, incluso el administrador.
- **RNF-6 Exstensibilidad:** la API del sistema debe ser fácilmente extensible a nuevas funcionalidades incorporando de manera eficaz soporte a nuevas peticiones.
- **RNF-7 Persistencia:** los servicios de procesamiento de las camas activas deben mantenerse funcionando aunque no existan clientes activos para evitar retrasos muy altos ante nuevas conexiones.
- **RNF-8 Fiabilidad:** los datos de la aplicación son correctos y actuales además de garantizar una predicción óptima del estado del paciente.

B.4. Especificación de requisitos

Los requisitos funcionales generan un conjunto de casos de uso que serán la base del desarrollo de la aplicación. La especificación de los mismos se encuentran entre la tabla B.1 y la tabla B.12. La representación gráfica se puede ver en los diagramas de las figuras B.1, **??**, B.3 y B.4. Existen dos actores, el **administrador** que se encarga de toda la labor de gestión tanto de usuarios como de camas y el **usuario** que únicamente puede gestionarse a si mismo y ver los datos de las camas que tenga permitidas.

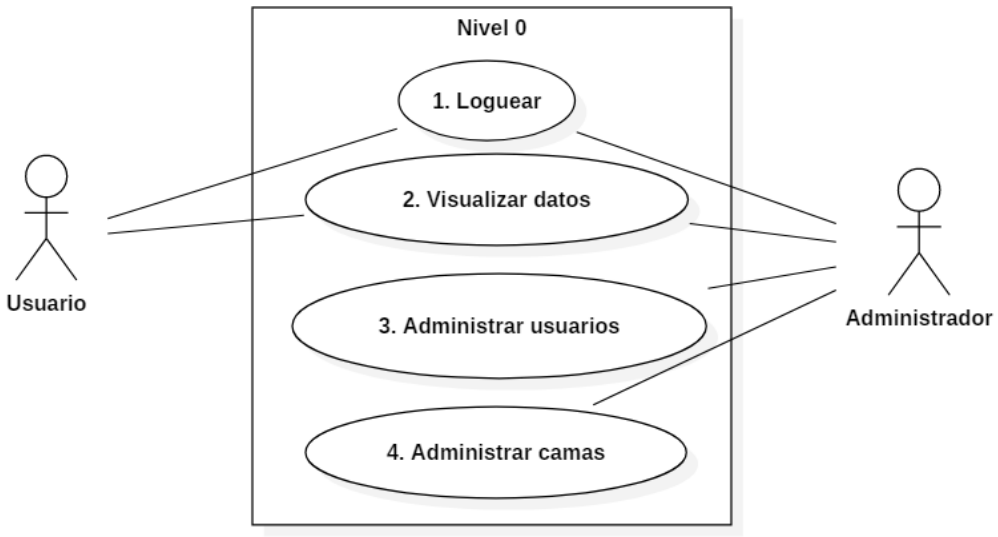

Figura B.1: Diagrama casos de uso - Nivel 0

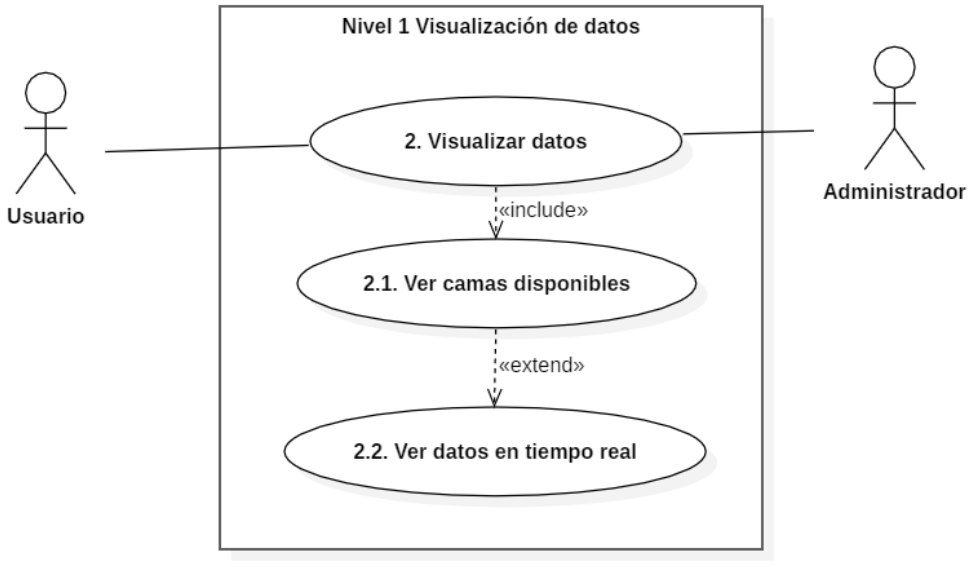

Figura B.2: Diagrama CU-2 - Nivel 1

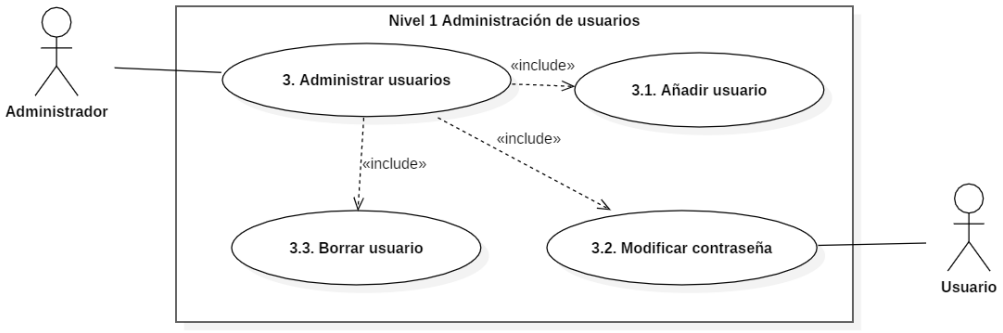

Figura B.3: Diagrama CU-3 - Nivel 1

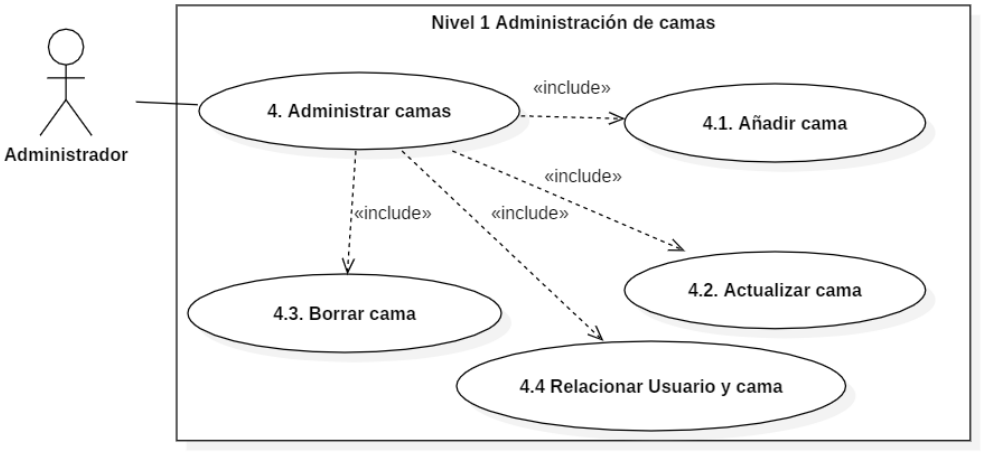

Figura B.4: Diagrama CU-4 - Nivel 1

CU-1: Loguear		
Descripción	El usuario se identifica en el sistema	
Precondiciones	No existe una sesión activa válida	
Requisitos	RF-1, RF-1.1	
Usuario	Anónimo	
Secuencia normal	Paso	Acción
	1	El cliente envía sus credenciales al servidor
	2	El servidor acepta las credenciales devolviendo el token de sesión
Postcondiciones	El usuario tiene una sesión activa válida	
Excepciones	Paso	Acción
	2	Si las credenciales son incorrectas el servidor responde con error
Frecuencia	Alta	
Importancia	Crítico	
Comentarios	Es siempre lo primero que aparecerá	

Tabla B.1: Caso de uso 1: Loguear

CU-2: Visualizar de datos		
Descripción	Ver lista de las camas disponibles	
Precondiciones	Sesión activa válida	
Requisitos	RF-1.3, RF-1.4	
Usuario	Administrador y Usuario	
Secuencia normal	Paso	Acción
	1	El cliente solicita ver las camas disponibles
Postcondiciones	El cliente está en la pantalla de camas disponibles	
Frecuencia	Alta	
Importancia	Alta	

Tabla B.2: Caso de uso 2: Visualizar de datos

CU-2.1: Elegir cama		
Descripción	Elegir cama	
Precondiciones	Sesión activa válida	
Requisitos	RF-1.3, RF-1.4, RF-4	
Usuario	Logueado	
Secuencia normal	Paso	Acción
	1	El cliente solicita ver las camas disponibles
	2	El servidor abre conexiones paralelas para actualizar en tiempo real el estado de las camas
	3	El cliente decide que cama ver
Postcondiciones	El cliente entra en la ventana de los datos en tiempo real	
Frecuencia	Alta	
Importancia	Alta	

Tabla B.3: Caso de uso 2.1: Elegir cama

CU-2.2: Ver datos en tiempo real		
Descripción	Ver datos en tiempo real	
Precondiciones	Sesión activa válida y cama existente y accesible	
Requisitos	RF-1.3, RF-1.4, RF-4	
Usuario	Administrador y usuario	
Secuencia normal	Paso	Acción
	1	El cliente solicita una nueva conexión
	2	El servidor provee una conexión en tiempo real con los datos
Postcondiciones	El usuario tiene una conexión paralela abierta con los datos en tiempo real	
Excepciones	Paso	Acción
	2	Si un paquete faltase o la señal fuera, débil se alertaría al usuario
Frecuencia	Alta	
Importancia	Máxima	

Tabla B.4: Caso de uso 2.2: Ver datos en tiempo real

CU-3: Administrar de usuarios		
Descripción	Administración de usuario: alta, baja y modificación	
Precondiciones	Sesión de administrador válida	
Requisitos	RF-3	
Usuario	Administrador	
Secuencia normal	Paso	Acción
	1	El administrador entra en el menú de administración de usuarios
Postcondiciones	El administrador está en el menú de administración de usuarios	
Frecuencia	Baja	
Importancia	Alta	

Tabla B.5: Caso de uso 3: Administrar de usuarios

CU-3.1: Añadir usuarios		
Descripción	Añadir usuarios	
Precondiciones	Sesión de administración activa	
Requisitos	RF-3.1	
Usuario	Administrador	
Secuencia normal	Paso	Acción
	1	El administrador elige añadir un nuevo usuario
	2	Se introduce un nombre de usuario para identificarlo
	3	Se introduce una contraseña dos veces
	4	Se almacenan los datos
Postcondiciones	Existe un nuevo usuario en el sistema	
Excepciones	Paso	Acción
	2	Si el nickname existiese
	3	La contraseña añadida no coincide en las dos ocasiones
Frecuencia	Baja	
Importancia	Alta	

Tabla B.6: Caso de uso 3.1: Añadir usuarios

CU-3.2: Modificar contraseña		
Descripción	Cambiar la contraseña de un usuario	
Precondiciones	Sesión activa válida, usuario existente	
Requisitos	RF-3.2	
Usuario	Administrador y Usuario	
Secuencia normal	Paso	Acción
	1	Si es usuario normal ir a 3
	2	Si es administrador elegir a qué usuario cambiar la contraseña
	3	Se introduce una contraseña nueva dos veces
	4	Se actualizan los datos
Postcondiciones	La contraseña ha cambiado	
Excepciones	Paso	Acción
	3	La contraseña añadida no coincide en las dos ocasiones
Frecuencia	Baja	
Importancia	Alta	

Tabla B.7: Caso de uso 3.2: Modificar contraseña

CU-3.3: Borrar usuario		
Descripción	Elimina un usuario de la base de datos	
Precondiciones	Sesión de administración válida, usuario existente	
Requisitos	RF-3.3	
Usuario	Administrador	
Secuencia normal	Paso	Acción
	1	Elegir a que usuario (no administrador) eliminar
	2	Eliminar usuario y todos los datos vinculados
Postcondiciones	El usuario ha sido eliminado	
Frecuencia	Baja	
Importancia	Media	

Tabla B.8: Caso de uso 3.3: Borrar usuario

CU-4: Administrar de camas		
Descripción	Administración de camas: alta, baja, modificación y asignación a usuarios	
Precondiciones	Sesión de administración válida	
Requisitos	RF-2	
Usuario	Administrador	
Secuencia normal	Paso	Acción
	1	El administrador entra en el menú de administración de camas
Postcondiciones	El administrador está en el menú de administración de camas	
Frecuencia	Baja	
Importancia	Media	

Tabla B.9: Caso de uso 4: Administrar de camas

CU-4.1: Añadir cama		
Descripción	Añadir cama	
Precondiciones	Sesión de administración válida	
Requisitos	RF-2.1	
Usuario	Administrador	
Secuencia normal	Paso	Acción
	1	El administrador elige añadir una nueva cama
	2	Se introduce el grupo multicast de la cama (IP y Puerto)
	3	Se introduce el nombre identificador
	4	Se almacenan los datos
Postcondiciones	Existe una nueva cama en el sistema	
Excepciones	Paso	Acción
	2	El grupo multicast pertenece a otra cama
	3	El nombre identificativo existe para otra cama
Frecuencia	Media	
Importancia	Crítica	
Comentarios	El grupo multicast se configura en la cama y el administrador solamente debe conocerlo, no configurar la cama física	

Tabla B.10: Caso de uso 4.1: Añadir cama

CU-2.2: Modificar cama		
Descripción	Modificar los datos de la cama	
Precondiciones	Sesión de administración válida, cama existente	
Requisitos	RF-2.2	
Usuario	Administrador	
Secuencia normal	Paso	Acción
	1	Se elige que cama modificar
	2	Se actualizan los datos a conveniencia del administrador según CU-4.1
	4	Se actualizan los datos
Postcondiciones	Los datos de la cama se modifican	
Excepciones	Paso	Acción
	2	Mismas excepciones que en CU-4.1
Frecuencia	Baja	
Importancia	Alta	

Tabla B.11: Caso de uso 4.2: Modificar cama

CU-4.3: Borrar cama		
Descripción	Elimina una cama de la base de datos	
Precondiciones	Sesión de administrador válida, cama existente	
Requisitos	RF-2.3	
Usuario	Administrador	
Secuencia normal	Paso	Acción
	1	Elegir a que cama eliminar
	2	Eliminar cama y todos los datos vinculados
Postcondiciones	La cama ya no está en la base de datos	
Frecuencia	Baja	
Importancia	Media	

Tabla B.12: Caso de uso 4.3: Borrar cama

CU-4.4: Asignar cama a usuario		
Descripción	Permite a un usuario ver los datos de una cama o quitar ese permiso	
Precondiciones	Sesión de administración válida, cama y usuario existentes	
Requisitos	RF-2.4	
Usuario	Administrador	
Secuencia normal	Paso	Acción
	1	Elegir cama
	2	Elegir usuario
	3	Si la relación existe se puede eliminar el permiso
	3	Si la relación no existe se puede crear el permiso
Postcondiciones	El usuario tiene acceso a la cama, o pierde el mismo	
Frecuencia	Media	
Importancia	Crítica	

Tabla B.13: Caso de uso 4.4: Asignar cama a usuario

Apéndice C

Especificación de diseño

C.1. Introducción

Tras el estudio de los requisitos se pasa a un diseño, en este apéndice se explicarán los distintos diseños de la aplicación, desde los datos, los procesos más importantes, el diseño arquitectónico y las interfaces.

C.2. Diseño de datos

De los requisitos y casos de usos se deduce el diseño de los datos. Para poder cumplir correctamente con las necesidades del cliente se introducen dos entidades, los **usuario** y las **camas** en la que cada uno almacena la información relevante propia. A su vez se relacionan en el sistema de permisos de qué persona puede ver qué cama. El diagrama Entidad-Relación se puede observar en la figura C.1.

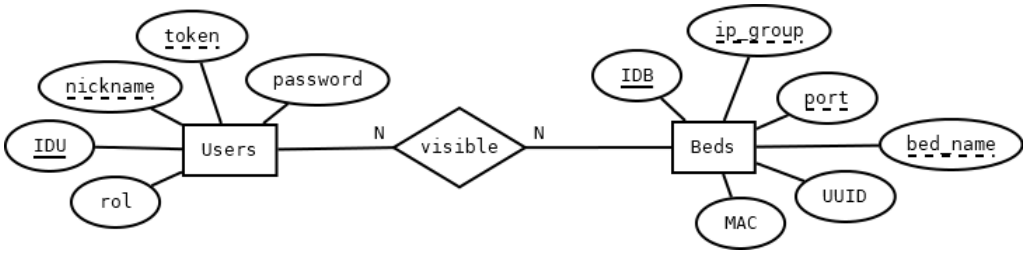

Figura C.1: Diagrama entidad-relación

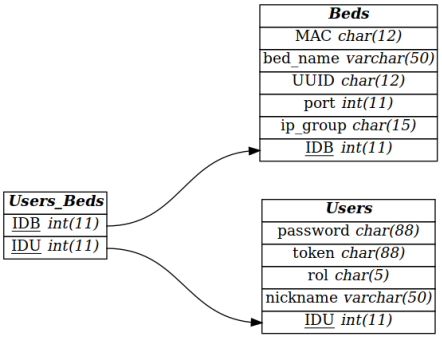

Figura C.2: Diagrama relacional

De este diagrama se genera el diagrama relacional (figura C.2) en el que se especifican las tablas que se usarán en el producto final.

C.3. Diseño procedimental

Existen diversos procesos importantes en la aplicación. El primero es la creación de los componentes que se encargan de monitorizar las camas, como se puede observar en la figura C.3. Al principio, el servidor instancia todos los *BedListener* necesarios según las camas que existan, y estos instancian su *BedProcess* correspondiente. Es importante destacar que estas dos clases funcionan como hilos.

Por otra parte el cliente abre conexiones con el servidor solicitando datos, este genera un hilo del *BedBroadcaster* que existe mientras el cliente esté activo (el que comenzó la petición o cualquiera que se incorpore posteriormente).

El proceso completo de la conexión del cliente se puede ver en la figura C.4. En esta cada llamada es un **evento** de *socketio* y los argumentos los datos que espera.

Los procedimiento de toda la *API* se especifican más adelante en la sección D.3.

C.4. Diseño arquitectónico

La arquitectura software sigue el patrón *MVC* [3]. Sin embargo, al ser un problema sencillo no se traslada a un diseño literal de este patrón. El

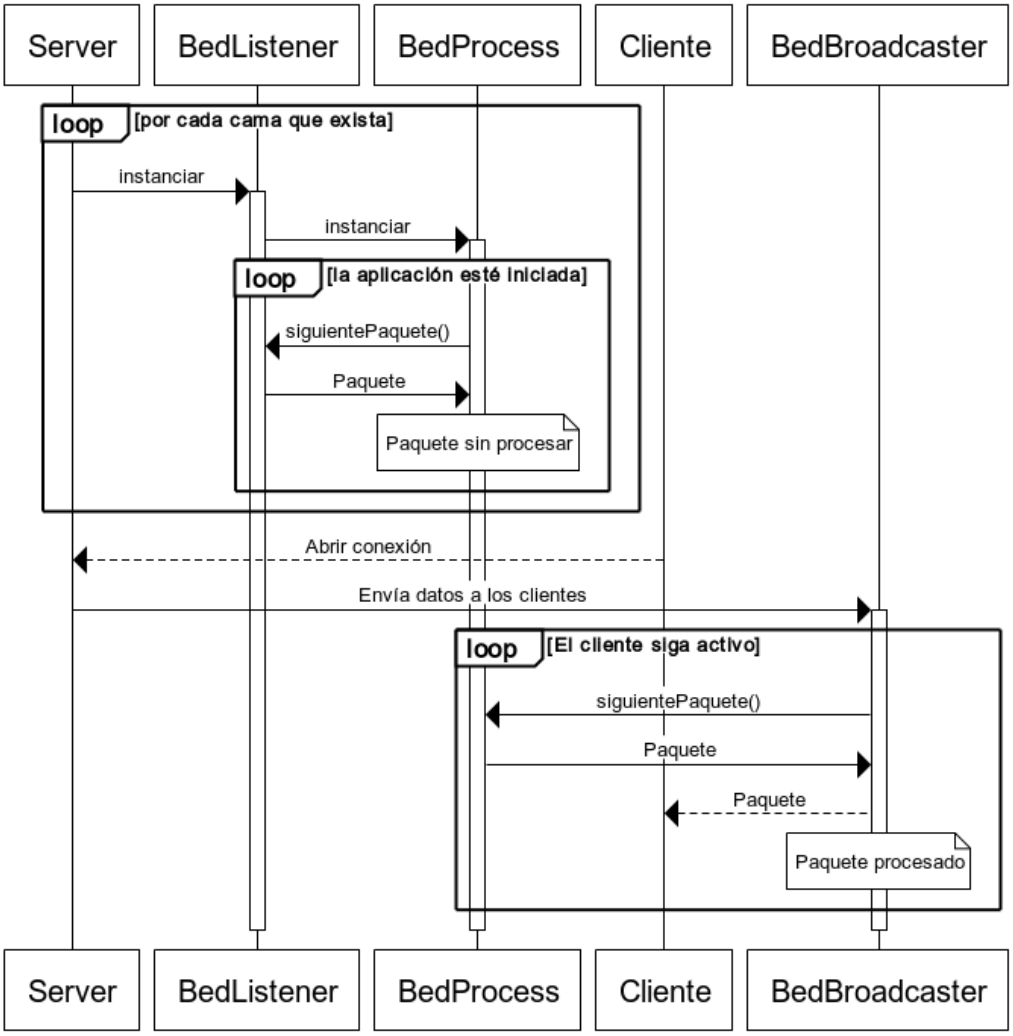

Figura C.3: Diagrama de secuencia de la creación de camas y la petición de paquetes

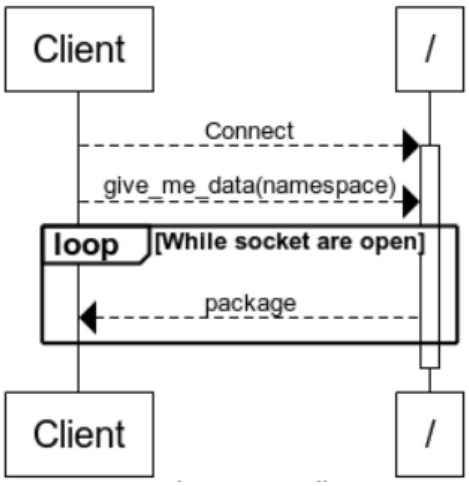

Figura C.4: Comunicación cliente servidor *websocket*

modelo se mantiene en la base de datos y no tiene una representación en ninguna clase. El *controlador* está pensado en varios componentes, una *API* que gestiona todos los accesos a la base de datos y un *proxy* que recoge las peticiones web y las preprocesa para redirigirlas a la *API*. Por último, la *vista* puede ser completamente diseñada para la situación que se requiera, en esta aplicación se gestiona mediante plantillas *HTML* y peticiones *AJAX* ya que se trata de una aplicación web. Gracias a que el controlador funciona como una *API REST* esta vista puede ser cambiada según se desee.

Otra arquitectura relevante es el *pipeline* de procesamiento de los datos de pacientes. Se puede observar, junto con los módulo de *routing*, en la figura C.5. Se trata de tres componentes, dos con contexto en toda la aplicación, que se trata del *listener* de la cama y el procesador de la misma. El último es el *broadcaster* que emite los datos a los pacientes. En la sección C.3 se puede ver la comunicación del cliente y el proceso de inicialización de los componentes de este *pipeline*.

La arquitectura global de la aplicación sigue el diagrama de componentes de la figura C.6. La aplicación, desarrollada en la librería de *Python*, *Flask*, se conecta a una base de datos relacional compatible con la API de *MySQL*, en el caso concreto del despliegue actual se usa la implementación de *MariaDB*; la aplicación también lanza hilos para la escucha de camas mediante multidifusión, aunque el diagrama solo tiene una pueden ser tantas como se requiera. El último módulo es el servidor web, en el caso del despliegue actual es un *Nginx* pero puede ser cualquiera, lo que debe ser de proxy reverso.

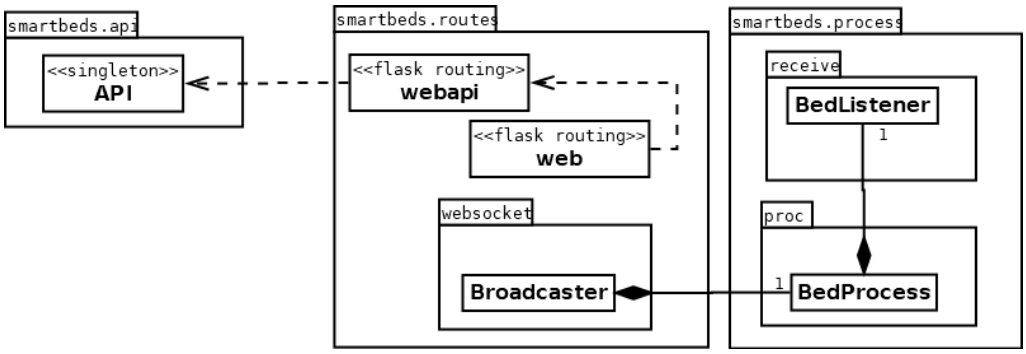

Figura C.5: Diagrama de clases

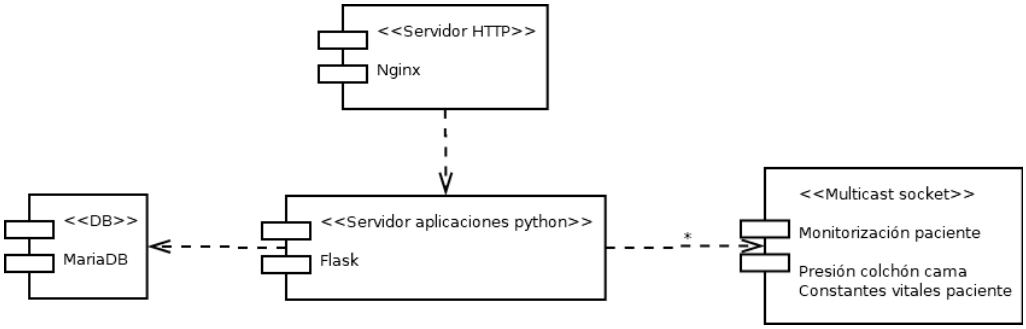

Figura C.6: Diagrama de despliegue

C.5. Diseño de interfaces

Las interfaces se han diseñado teniendo en cuenta la usabilidad de la misma, así como un diseño simple que permitiese que fuese más intuitiva con una curva de aprendizaje leve. Se ha tenido en cuenta también que la interfaz sea adaptable a una gran variedad de pantallas teniendo en cuenta que, al ser una aplicación web, el uso de la misma puede ser en multitud de dispositivos diferentes (diseño *responsive* [4]).

Los primeros prototipos fueron realizados sobre el caso de uso especificado en la tabla B.4, tanto para escritorio (imagen C.7) como para móvil (imagen C.8).

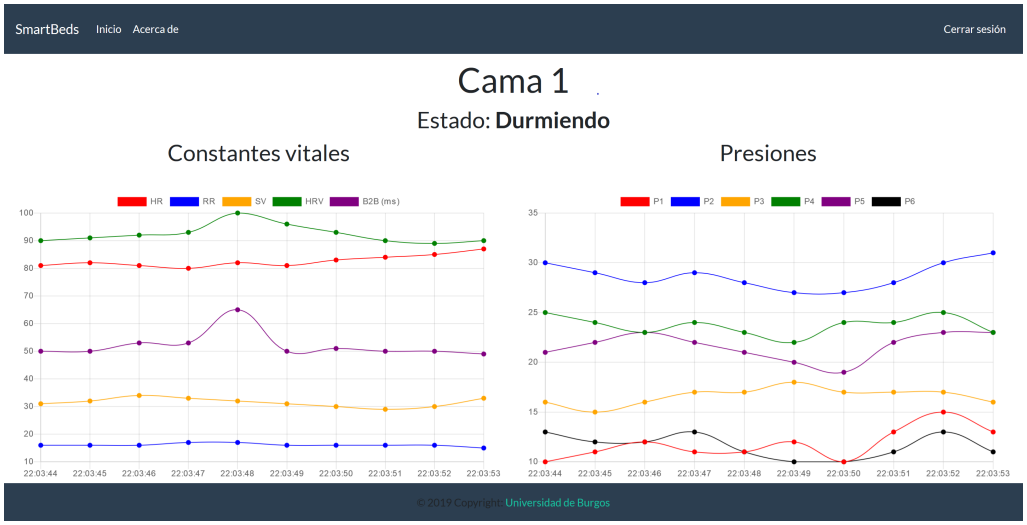

Figura C.7: Prototipo para escritorio

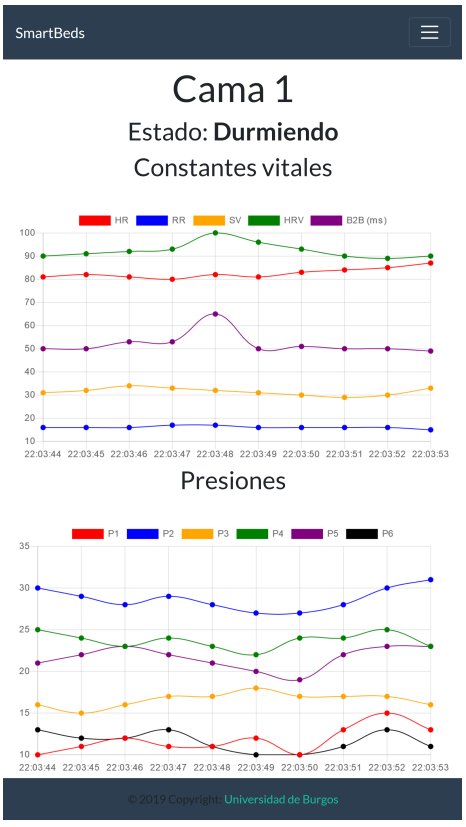

Figura C.8: Prototipo para móvil

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apéndice se explicarán todos los detalles necesarios para poder entender los componentes software que existen así como como desarrollar aplicaciones siguiendo la *API REST* provista.

D.2. Estructura de directorios

En la carpeta `doc` se encuentra toda la documentación de este proyecto. En `research` están los ficheros de la investigación, dentro de esta están `data`[1] que contiene los datos proporcionados para realizar el estudio, `model` con los modelos binarios, `src` con los ficheros fuentes y los *Jupyter Notebooks* con la investigación y en `tests` con los tests sobre algunas fuentes.

La carpeta `smartebs` están las fuentes de la aplicación web siendo el modulo raíz. Dentro de esta están los módulo `api` con la `API` web, `process` con los algoritmos de procesamiento de los datos, `resources` con la configuración y los `assets` de la página web (plantillas, códigos `JavaScript` etc), `routes` que engloba las rutas *URI* para dirigir las peticiones. Por último, está la carpeta `test` con las pruebas unitarias.

[1] Esta carpeta no se encuentra en el repositorio *GitHub* porque contienen datos privados

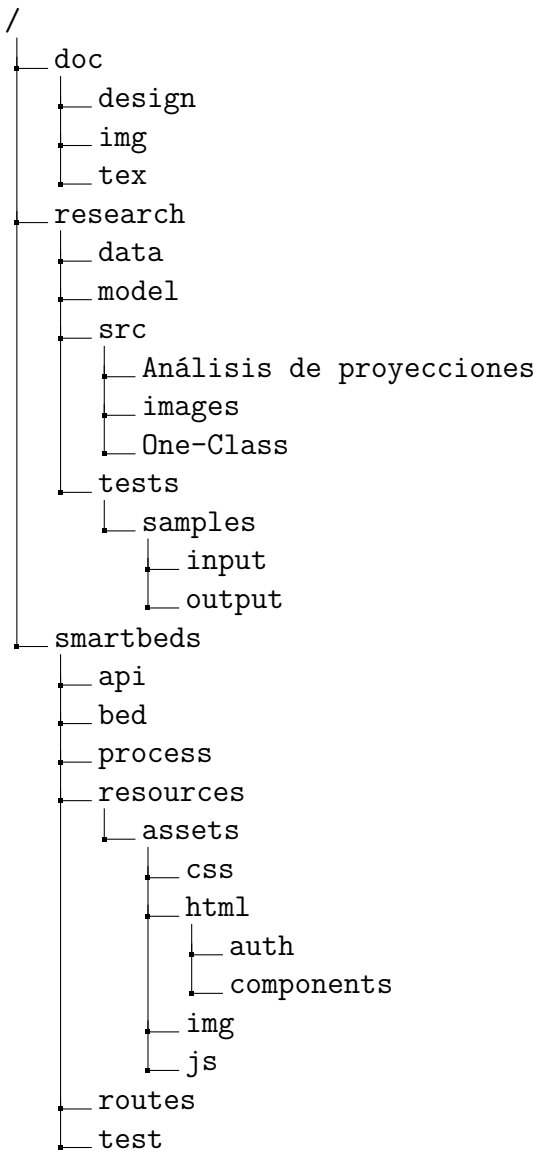

Figura D.1: Árbol de directorios

De manera especial se encuentra la carpeta `bed` dentro de `smartbeds` que contiene un *script* para la simulación de camas.

El árbol de directorios del proyecto se puede ver en la figura D.1

D.3. Manual del programador

API

Para proveer los servicios de esta aplicación a nuevos entornos se incorpora una API para utilizar los diferentes servicios del sistema especificados en los Casos de Uso, sección B.4.

El funcionamiento general de la API serán peticiones `POST` mediante `application/x-www-form-urlencoded` ante rutas específicas con los datos requeridos para cada petición. Y según sea el caso el servidor contestará con un fichero `JSON` con la respuesta adecuada. De manera particular está el sistema en tiempo real que funciona mediante mensajes de *WebSockets* [5] usando la librería *SocketIO* [1] mediante la serie de eventos que se pueden ver en la sección C.3 en la figura C.4

Todas las respuestas del servidor contendrán los siguientes campos

```
{
  "status": 200|400|401|403|404|418|500,
  "message": "OK"|"Error description"
}
```

El valor de `status` tendrá un valor según los códigos HTTP definidos en el RFC 7231 [2] y el mensaje será una explicación detallada del error producido.

Las distintas peticiones se especifican en la tabla D.1.

CU	URI	Petición	Respuesta
CU-1	`/api/auth`	`"user": text, "pass": text`	`{ ..., "token": text, "role": text, "username": text }`

continúa en la página siguiente

continúa desde la página anterior

CU	URI	Petición	Respuesta

CU-2.1
CU-4
	/api/beds	"token":
		text
			...,
			"beds": [{
			"bed_name":
			text,
			"ip_group":
			text,
			"port": text,
			"MAC": text,
			"UUID": text
			}
			...]

CU-2.2	/api/bed	token=text&
		bedname=
		text
			{
			...,
			"namespace": text
			}

CU-2.2	/ *(WebSocket)*	{
		"
		namespace
		":
		namespace
		}
			{
			"results": [result,
			prob,
			press_state,
			hr_state],
			"instance":
			datetime
			"vital": [HR,RR,SV,
			HRV,B2B],
			"pressure": [P1,P2,
			..,P6]
			}

continúa en la página siguiente

continúa desde la página anterior

CU	URI	Petición	Respuesta
CU-3	/api/users	token=text	{ ..., "users":[text,...,text] }
CU-3.1	/api/user/add	token=text& username=text& password=text& password-re=text	{ ... }
CU-3.2	/api/user/mod	token=text& username=text& password=text& password-re=text [&pasword-old=text]	{ ... }
CU-3.3	/api/user/del	token=text& username=text	{ ... }
CU-4.1	/api/bed/add	token=text& bed_name=text& ip_group=text& port=text& MAC=text& UUID=text	{ ... }

continúa en la página siguiente

continúa desde la página anterior

CU	URI	Petición	Respuesta

CU-4.2	/api/bed/mod	token=text&
		bed_name=
		text&
		ip_group=
		text&
		port=text&
		MAC=text&
		UUID=text
			{
			...
			}

CU-4.3	/api/bed/del	token=text&
		bed_name=
		text
			{
			...
			}

CU-4.4	/api/bed/perm	token=text&
		mode=[info|
		change]
		[&bed_name=
		text&
		username=
		text]
			{
			...,
			"permission":[
			{
			"username":text,
			"bed_name":text
			},
			...
			]
			}

Tabla D.1: Especificaciones del API

D.4. Compilación, instalación y ejecución del proyecto

Este proyecto contiene un servidor web, del cual se explicará su proceso de despliegue en este apartado. Sin embargo, no disponemos de camas reales que puedan aportar datos en tiempo real como hemos visto en la figura C.6 debido a que este *hardware* es privado.

En su lugar se ha creado un *script* de *Python* que emite cada 0,4 segundos de manera indefinida datos que se han obtenido previamente. Este *script* se encuentra en la carpeta `smartbeds/bed`.

Requisitos del sistema

Para el funcionamiento correcto de la aplicación se requiere de *hardware* los siguientes mínimos:

- **Procesador**: arquitectura de 64 bits con soporte multihilo. Se recomienda un mínimo de 1,5 GHz, 16MiB de memoria caché y dos núcleos.
- **Memoria**: 768MiB, incorporando 256MiB aprox. por cada nueva cama incorporada.
- **Almacenamiento**: 16MiB aprox.

Requisitos software

Para poder ejecutar la aplicación se necesitan los siguientes apartados de software.

Entorno *Python*, versión 3.7 con las librerías de *requeriments.txt* instaladas (`pip install -r requeriments.txt`). Base de datos *MySQL* o con API compatible como *MariaDB*. Servidor web con soporte de *proxy* reverso y *websokets* como *Nginx* versión 1.16.

Instalación en entorno GNU/Linux

El primer paso necesario es la configuración de la base de datos en la cual se van a almacenar los datos del sistema. La estructura de esta base de datos se encuentra en el fichero `/smartbeds/resources/database.sql`. Tras esto es necesario crear el fichero `/smartbeds/resources/project.json` que tendrá la configuración de la aplicación siguiendo este formato:

```
{
        "secret-key": "clave secreta",
        "url":  "http://127.0.0.1",
        "port": 3031,
        "mode": "ssl"
        "database": {
                "host": "database-host",
                "database": "database-name",
                "user": "database-user",
```

```
10                  "password": "database-password"
11          }
12  }
```

En cada campo habría que incorporar los cambios oportunos y se recomienda no modificar el campo `url`. De manera especial el campo `mode` puede ser `ssl` o `no-ssl` si los *websockets* se ejecutan sobre *SSL* o no. Tras esto ya el programa es ejecutable y puede funcionar al ejecutar el fichero `/index.py`.

El siguiente paso es crear la configuración del servidor *HTTP* para redirigir las peticiones externas a la aplicación local. Como ejemplo, para un servidor *Nginx* la configuración sería la siguiente:

```
1  server {
2          listen 443 ssl http2;
3          server_name $server_name$;
4          root $route$;
5
6          location / {
7                  include proxy_params;
8                  proxy_pass http://127.0.0.1:3031;
9          }
10
11         location /socket.io {
12                 include proxy_params;
13                 proxy_http_version 1.1;
14                 proxy_buffering off;
15                 proxy_set_header Upgrade
                      $http_upgrade;
16                 proxy_set_header Connection "
                      Upgrade";
17                 proxy_pass http://127.0.0.1:3031/
                      socket.io;
18         }
19 }
```

Siendo necesario modificar el nombre del servidor por el dominio o IP de acceso y la raíz como la ruta donde se encuentra el fichero `index.py`, aunque

esto último no es necesario para el funcionamiento, solo como referencia interna.

Por último es necesario crear un demonio de `systemd` que ejecute la aplicación en segundo plano. El fichero de configuración de este demonio se alojaría en `/usr/lib/systemd/system/smartbeds.service`. Siendo el fichero en cuestión semejante a:

```
[Unit]
Description=Smartbed Service
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=http
WorkingDirectory=/ruta/a/la/carpeta
ExecStart=python index.py

[Install]
WantedBy=multi-user.target
```

El campo `ExecStart` podría cambiar en el caso de utilizar algún entornos *Python* como son los que provee conda. En el caso de ser así se recomienda cambiar el campo por `bash index.sh` siendo el fichero semejante a:

```
#!/bin/bash
/opt/miniconda3/envs/entorno/bin/python index.py
```

Finalmente solo hace falta lanzar el demonio con el comando `systemctl start smartbeds`.

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

En este apartado se va a explicar como se usa la aplicación para los usuarios posibles. Estos se dividen en dos roles, los administradores y los usuarios.

E.2. Requisitos de usuarios

Todos los tipos de usuario requerirán tener un navegador compatible con `HTML 5` y conexión a Internet.

E.3. Instalación

Al tratarse de una aplicación web no se requiere una instalación.

E.4. Manual del usuario

Bibliografía

[1] Damien Arrachequesne and Erik Little. Socket.io. https://socket.io/docs/, feb 2018.

[2] R. Fielding and J. Reschke. Hypertext transfer protocol (http/1.1): Semantics and content. RFC 7231, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7231.txt.

[3] Wikipedia contributors. Model–view–controller — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=901347868, 2019. [Online; accessed 11-June-2019].

[4] Wikipedia contributors. Responsive web design — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Responsive_web_design&oldid=899574617, 2019. [Online; accessed 31-May-2019].

[5] Wikipedia contributors. Websocket — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=WebSocket&oldid=897985416, 2019. [Online; accessed 31-May-2019].

[6] Zhenyue Zhang and Hongyuan Zha. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM journal on scientific computing*, 26(1):313–338, 2004.