ANEXO IV: Diseño del sistema software

Desarrollo de una plataforma o aplicación web para la gestión de un gimnasio

Trabajo fin de grado
Grado en Ingeniería Informática

VNiVERSiDAD D SALAMANCA

Julio de 2021

Autor

Alberto Martín Peralejo

Tutores

André Filipe Sales Mendes
Gabriel Villarrubia González
Juan Francisco de Paz Santana

Índice general

1. Introducción **1**

2. Modelo de diseño **2**
2.1. Patrones arquitectónicos . . . 2
2.1.1. Modelo-vista-modelo de vista (MVVM) . . . 2
2.1.1.1. MVVM en Vue.js: . . . 4
2.1.2. Singleton . . . 4
2.1.3. State management . . . 5
2.2. Subsistemas de diseño . . . 6
2.3. Clases de diseño . . . 7
2.3.1. Frontend . . . 7
2.3.1.1. Vista . . . 7
2.3.1.2. Modelo de Vista . . . 8
2.3.1.3. Modelo . . . 9
2.3.2. Backend . . . 9
2.3.2.1. Router . . . 9
2.3.2.2. Middlewares . . . 10
2.3.2.3. Controllers . . . 11
2.3.2.4. Models . . . 12
2.3.2.5. Server . . . 12
2.3.3. Base de datos . . . 13
2.4. Vista arquitectónica . . . 14
2.5. Realización de casos de uso . . . 14
2.5.1. Gestión de autentificación . . . 15
2.5.2. Gestión de matrículas . . . 19
2.5.3. Gestión de usuario . . . 22
2.5.4. Gestión de actividades . . . 28
2.5.5. Gestión de datos de usuario . . . 36
2.5.6. Gestión de gimnasio . . . 38
2.5.7. Gestión de pádel . . . 45

3. Diseño de la base de datos **50**

4. Modelo de despliegue **52**

5. Bibliografía **54**

Índice de figuras

2.1. Patrón MVVM . . . 3
2.2. Patrón MVVM en Vue.js . . . 4
2.3. Patrón Singleton . . . 5
2.4. Patrón State management . . . 6
2.5. Subsistemas de diseño . . . 7
2.6. Vista . . . 8
2.7. Modelo de Vista . . . 8
2.8. Modelo . . . 9
2.9. Router . . . 9
2.10. Middlewares . . . 10
2.11. Controllers . . . 11
2.12. Models . . . 12
2.13. Server . . . 12
2.14. Base de datos . . . 13
2.15. Diagrama de Clases de las relaciones de la base de datos . . . 13
2.16. Vista arquitectónica del sistema . . . 14
2.17. UC-0001 Iniciar sesión . . . 15
2.18. UC-0002 Registrarse . . . 16
2.19. UC-0003 Solicitar restablecer contraseña . . . 17
2.20. UC-0004 Resetear contraseña . . . 18
2.21. UC-0005 Cerrar sesión . . . 18
2.22. UC-0006 Listar matrículas . . . 19
2.23. UC-0027 Pagar . . . 20
2.24. UC-0028 Actualizar matrículas . . . 21
2.25. UC-0029 Comprobar pago . . . 21
2.26. UC-0007 Ver ubicación . . . 22
2.27. UC-0008 Ver información . . . 22
2.28. UC-0051 Actualizar Nºreservas semanales . . . 23
2.29. UC-0052 Listar usuarios . . . 23
2.30. UC-0053 Buscar usuario . . . 24
2.31. UC-0054 Borrar usuario . . . 24
2.32. UC-0055 Ver usuario . . . 25
2.33. UC-0056 Cancelar reserva de usuario . . . 26
2.34. UC-0057 Hacer administrador . . . 27
2.35. UC-0009 Listar actividades . . . 28
2.36. UC-0010 Buscar actividad . . . 28
2.37. UC-0011 Ver actividad . . . 29
2.38. UC-0012 Apuntar actividad . . . 29
2.39. UC-0013 Desapuntar actividad . . . 30
2.40. UC-0014 Escribir reseña . . . 30
2.41. UC-0015 Borrar reseña . . . 31
2.42. UC-0016 Votar reseña . . . 31
2.43. UC-0017 Compartir en las redes . . . 32

2.44. UC-0018 Listar reseñas 32
2.45. UC-0019 Borrar actividad 33
2.46. UC-0020 Subir imagen 34
2.47. UC-0021 Borrar imagen 34
2.48. UC-0022 Crear actividad 35
2.49. UC-0023 Listar datos personales 36
2.50. UC-0024 Actualizar datos personales 36
2.51. UC-0025 Subir imagen personal 37
2.52. UC-0026 Borrar imagen personal 37
2.53. UC-0030 Ver gimnasio 38
2.54. UC-0031 Listar máquinas 38
2.55. UC-0032 Ver día gimnasio 39
2.56. UC-0033 Reservar hora 40
2.57. UC-0034 Listar reservas 41
2.58. UC-0035 Cancelar reservas 41
2.59. UC-0041 Registrar máquina 42
2.60. UC-0042 Borrar máquina 42
2.61. UC-0043 Comprobar usuario tiene hora 43
2.62. UC-0045 Actualizar aforo gimnasio 43
2.63. UC-0047 Borrar reservas canceladas gimnasio 44
2.64. UC-0049 Borrar reservas caducadas gimnasio 44
2.65. UC-0036 Ver pádel 45
2.66. UC-0037 Ver día pádel 45
2.67. UC-0038 Reservar hora pádel 46
2.68. UC-0039 Listar reservas pádel 46
2.69. UC-0040 Cancelar reservas pádel 47
2.70. UC-0044 Comprobar usuario tiene hora pádel 47
2.71. UC-0046 Actualizar estado pádel 48
2.72. UC-0048 Borrar reservas canceladas pádel 48
2.73. UC-0050 Borrar reservas caducadas pádel 49

3.1. Ejemplo JSON de un usuario registrado 50

4.1. Diagrama de despliegue 52

1. Introducción

Este anexo tiene como propósito recoger la documentación de la fase de diseño del sistema. Mientras la fase de Análisis se basaba en el dominio del problema, la de diseño se centra en el dominio de la solución [6] [5].

Por lo tanto, como se centra en el dominio de la solución los nombres de clases, métodos y atributos que se mostrarán a lo largo de este documento serán una aproximación cercana a la que nos encontremos en la implementación, es decir, se trata de una aproximación cercana a la implementación.

Para la generación de los diagramas mostrados en este documento se ha utilizado la herramienta UML Visual Paradigm [2].

La estructura del anexo es la siguiente:

- **Modelo de diseño:**
  - **Patrones arquitectónicos:** Se detallan los patrones arquitectónicos que serán utilizados durante el diseño.
  - **Subsistemas de diseño:** Se organiza el sistema en paquetes y se muestra la relación entre ellos.
  - **Clases de diseño:** Se enseñan las clases de diseño, con sus atributos, operaciones y relaciones.
  - **Vista arquitectónica:** Vista detallada a partir de las clases de diseño las distintas capas del patrón Modelo-Vista-ModeloVista de la arquitectura diseñada.
  - **Realización de casos de uso:** Se podrán observar los diagramas de secuencia de diseño de los casos de uso del sistema.
- **Diseño de la base de datos:** Mediante un diagrama Entidad-relación se muestra la información que maneja el sistema.
- **Modelo de despliegue:** Detalla el despliegue de los nodos y artefactos del sistema.

2. Modelo de diseño

El modelo de diseño es una abstracción de la implementación del sistema. Se utiliza para concebir y para documentar el diseño del sistema de software. Es un producto de trabajo integral y compuesto que abarca todas las clases de diseño, subsistemas, paquetes, colaboraciones y las relaciones entre ellos.

En esta fase del desarrollo se ha decidido utilizar varios patrones que nos facilitarán el desarrollo. El patrón arquitectónico Modelo Vista-Modelo de vista (MVVM) se utilizará para estructurar la arquitectura del sistema.

Se han utilizado las siguientes tecnologías:

- **Vue.js:** Framework de desarrollo web, basado en el patrón MVVM. Cabe destacar el uso de las librerías Vuex, VueRouter y Vuetify.
- **HTML y CSS:** Para el desarrollo de las vistas.
- **Node.js:** Como plataforma que permite programación del lado del servidor.
- **JavaScript:** Usado tanto en el Frontend como en el Backend como lenguaje de programación.
- **MongoDB:** Como base de datos del sistema.

2.1. Patrones arquitectónicos

Un patrón arquitectónico expresa un esquema de organización estructural fundamental para sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y directrices para organizar las relaciones entre ellos.

2.1.1. Modelo-vista-modelo de vista (MVVM)

MVVM se deriva del patrón clásico MVC (Modelo-Vista-Controlador). El patrón modelo–vista–modelo de vista (en inglés, model–view–viewmodel, abreviado MVVM) se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación, facilitando las pruebas, mantenimiento y la escalabilidad de los proyectos.

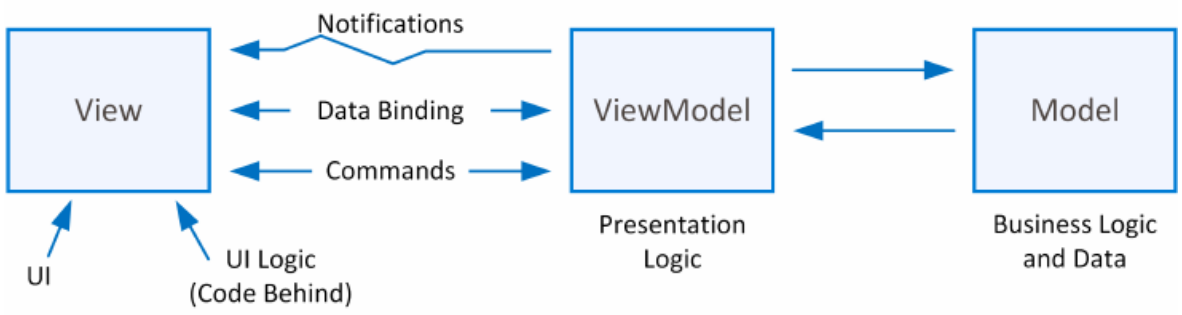

Figura 2.1: Patrón MVVM

Estos son los tres componentes que forman el patrón MVVM:

- **Modelo:** Representa la capa de datos y/o la lógica de negocio, también denominado como el objeto del dominio. El modelo contiene la información, pero nunca las acciones o servicios que la manipulan. En ningún caso tiene dependencia alguna con la vista.
- **Vista:** La misión de la vista es representar la información a través de los elementos visuales que la componen. Las vistas en MVVM son activas, contienen comportamientos, eventos y enlaces a datos que, en cierta manera, necesitan tener conocimiento del modelo subyacente.
- **Modelo de vista:** El modelo de vista es un actor intermediario entre el modelo y la vista, contiene toda la lógica de presentación y se comporta como una abstracción de la interfaz. La comunicación entre la vista y el viewmodel se realiza por medio los enlaces de datos (binders).

  El modelo de vista también se encarga de realizar la comunicación entre los modelos y las vistas.

Propagación Bi-direccional: O Two-Ways Data Binding, es un protocolo en el cual cualquier cambio producido en un objeto, este se propaga a todas las capas en las que se encuentra, manteniendo el estado de este consistente en los diferentes compontentes.

Ventajas del MVVM:

- **Acoplamiento bajo:** las vistas pueden ser independientes de los cambios y modificaciones del modelo. Un modelo de vista se puede vincular a diferentes vistas. Cuando la vista cambia, el modelo puede permanecer sin cambios, y cuando el modelo cambia, la vista también puede permanecer sin cambios.
- **Reutilizable:** puede poner algo de lógica de vista en un modelo de vista y permitir que muchas vistas reutilicen esta lógica de vista.
- **Desarrollo independiente:** los desarrolladores pueden centrarse en la lógica

empresarial y el desarrollo de datos (ViewModel), y los diseñadores pueden centrarse en el diseño de páginas.

- **Comprobable:** la interfaz siempre ha sido difícil de probar, pero ahora la prueba se puede escribir para ViewModel.

2.1.1.1. MVVM en Vue.js:

Vue.js, que es el marco JavaScript que se va a utilizar para implementar este proyecto, está orientado al patrón MVVM, centrándose en concreto en la parte de la Vista del modelo, pues conecta la Vista y el Modelo a través de data bindings [3].

Este framework web está diseñado para aplicarse capa por capa de abajo hacia arriba. La biblioteca principal de Vue solo se enfoca en la capa de vista, que no solo es fácil de usar, sino también fácil de integrar con bibliotecas de terceros (como: vue-router, vue-resource, vuex).

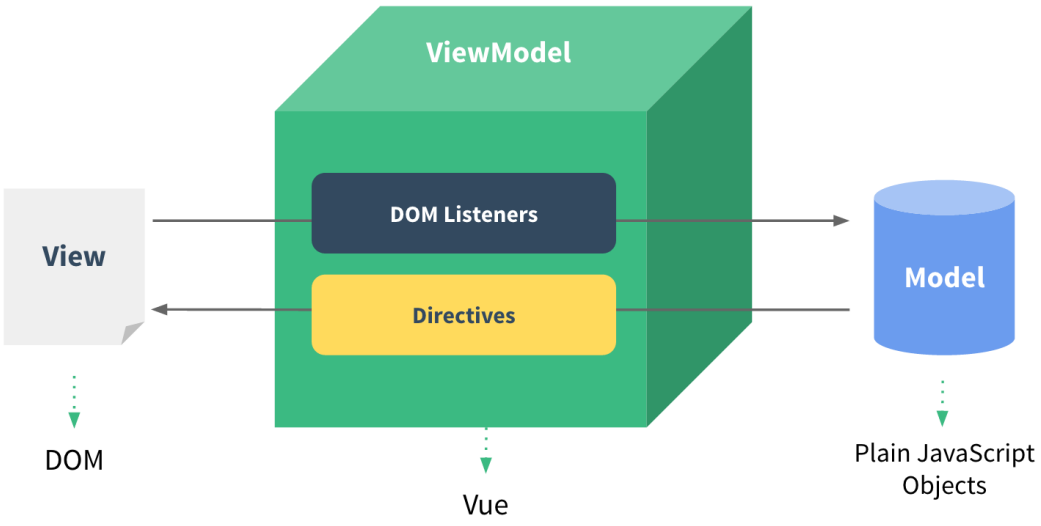

Figura 2.2: Patrón MVVM en Vue.js

Se puede decir que Vue.js es la mejor práctica de la arquitectura MVVM, centrándose en ViewModel en MVVM, no solo logra un enlace de datos bidireccional, sino también una biblioteca JS relativamente liviana, La API es concisa y fácil de usar.

2.1.2. Singleton

Singleton es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

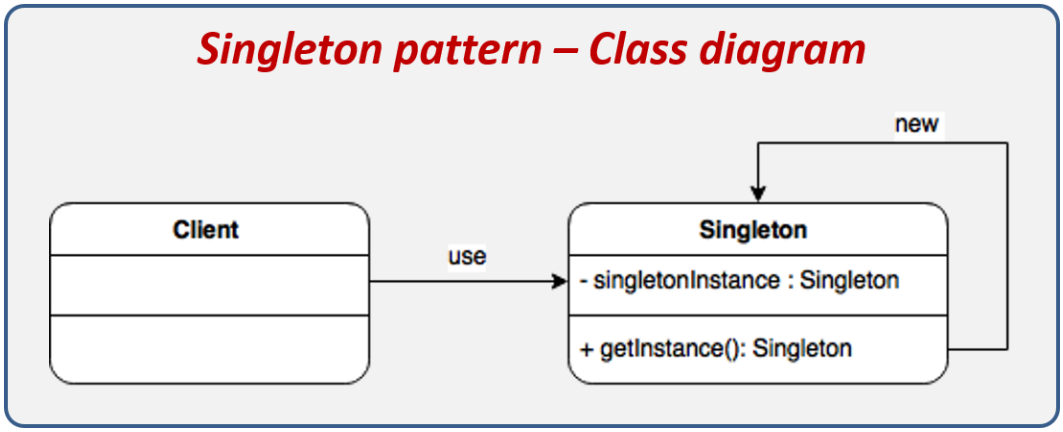

Figura 2.3: Patrón Singleton

Se utiliza de manera conjunta junto al patrón State management a través de la extensión Vuex de Vue.js. La instancia del objeto singleton creado es obtenida por los componentes a través del DOM.

2.1.3. State management

Patrón de administración del estado o State management es un patrón de comportamiento que permite establecer el acceso a objetos a través de Stores, de tal modo que distintos componentes puedan acceder a un estado compartido y así evitar información duplicada o tener que utilizar el árbol de componentes para la transmitisión de datos [4].

Una de las ventajas consiste en ofrecer contenido sincronizado y favorecer la reactividad entre los componentes en una web.

Además, las implementaciones de este patrón suelen favorecer un acceso a los datos ordenado y a través de una interfaz, como es el caso de Vuex, que ofrece las siguientes interfaces:

- **Actions** Un componente que accede al Store puede realizar cambios a través de métodos llamados actions, que son las funciones asíncronas encargadas de permitir la realización de campos a los elementos externos al Store.
- **Mutations:** Los métodos action realizan modificaciones sobre el estado mediante la acción commit, que llama a mutaciones, estas son funciones síncronas que modifican los datos de forma ordenada.
- **State:** Contiene los estados de los objetos que se desean compartir entre las vistas.

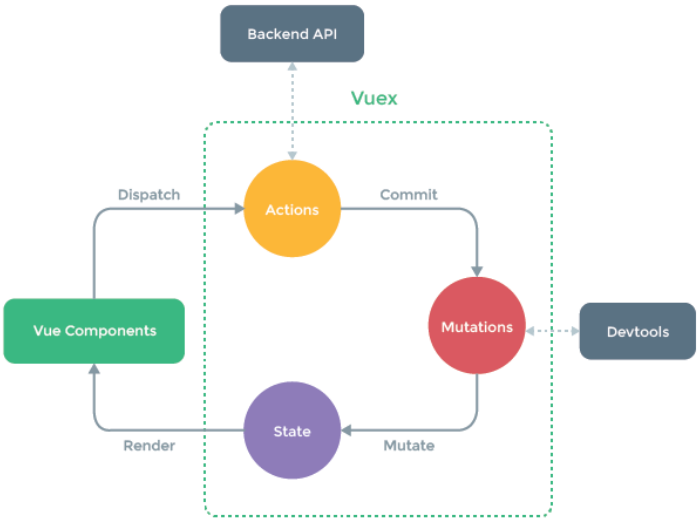

Figura 2.4: Patrón State management

Se ha utilizado este patrón a través de la extensión Vuex para vue.js, diseñada para este propósito. Diseñando un único store que carga la información necesaria de la base de datos.

2.2. Subsistemas de diseño

A partir de este punto del diseño, se ha desglosado el sistema en subpaquetes más específicos:

- **Paquete Frontend:** Paquete que ofrece el acceso al sistema a los usuarios. El sistema tiene diferentes interfaces dependiendo del rol del usuario. Sigue la estructura del patrón MVVM.
- **Paquete Backend:** Paquete encargado de procesar las peticiones de los usuarios y guardar los cambios de información en base de datos.
  - **Subpquete Router:** Subaquete donde se declaran las funciones que conforman la API Rest, estas pueden ser accedidas a través del protocolo HTTP.
  - **Subpaquete Middlewares:** Subpaquete que procesa las peticiones de los usuarios y controlar ciertas condiciones.
  - **Subpaquete Controllers:** Subpaquete encargado de procesar las peticiones de los usuarios cuando se hayan cumplido ciertas condiciones y guardar los cambios de información en base de datos.
  - **Subpaquete Models:** Subpaquete que representa la persistencia de la información.

- **Subpaquete Server:** Subpaquete que inicia la conexión con la base de datos y la escucha de peticiones.

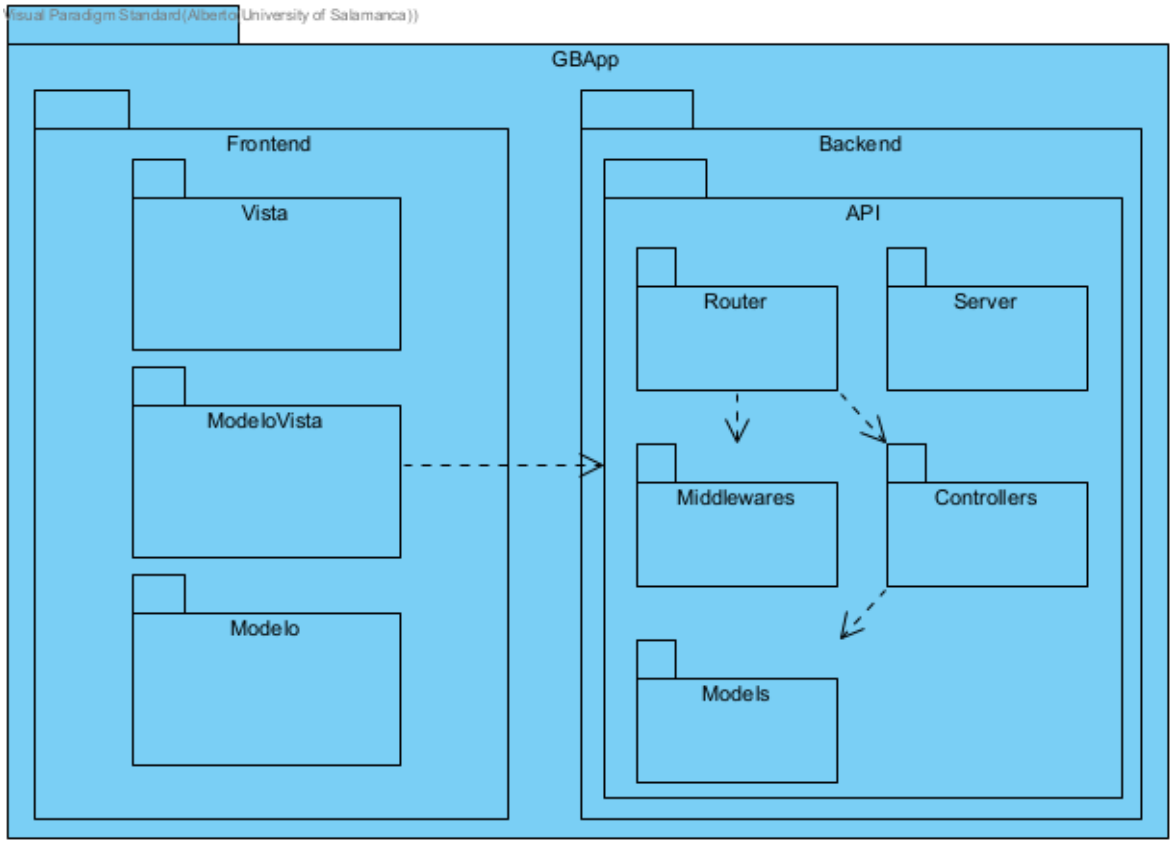

Figura 2.5: Subsistemas de diseño

2.3. Clases de diseño

En esta sección, se describen los paquetes de los subsistemas y las clases de diseño que forman.

2.3.1. Frontend

2.3.1.1. Vista

En la siguiente figura se pueden observar los distintos componentes que forman la vista de la aplicación web.

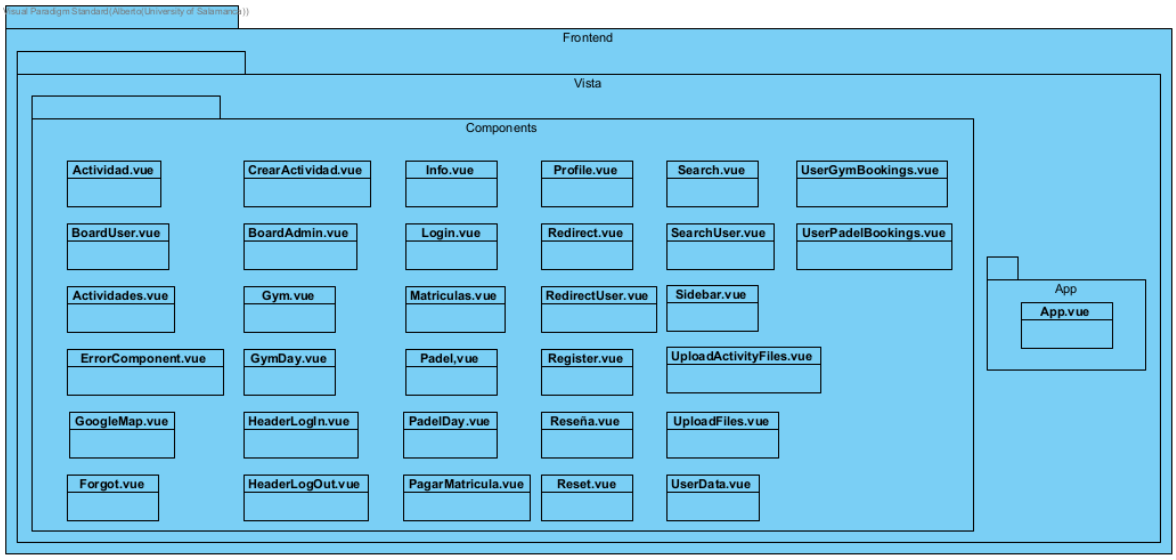

Figura 2.6: Vista

2.3.1.2. Modelo de Vista

En la siguiente figura se especifican los diferentes modelos de vista, que gestionan el flujo de datos entre el modelo y la vista a través de métodos, eventos y especificaciones del ciclo de vida establecido por Vue.js.

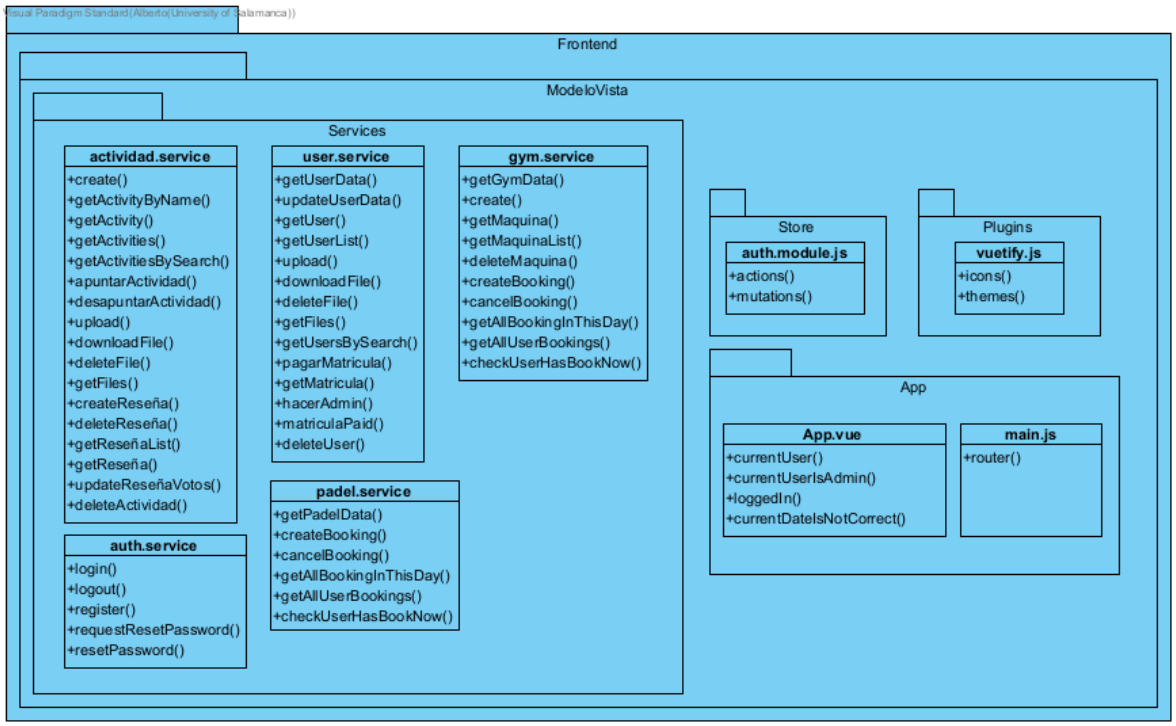

Figura 2.7: Modelo de Vista

2.3.1.3. Modelo

En la siguiente figura se especifican los modelos de los que hará uso la aplicación web, como estamos tratando con un MVVM basado en javascript en el que no existen clases y los datos son tratados desde el Modelo de Vista solo disponen de atributos.

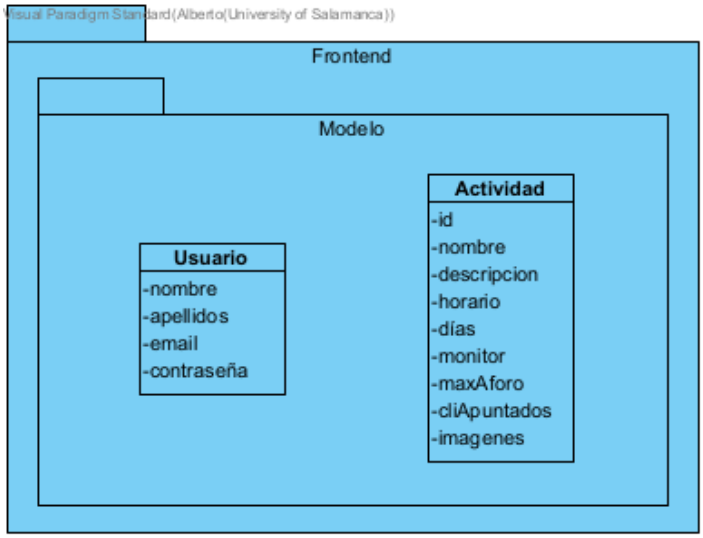

Figura 2.8: Modelo

2.3.2. Backend

2.3.2.1. Router

Redirige las peticiones a los middlewares y controllers.

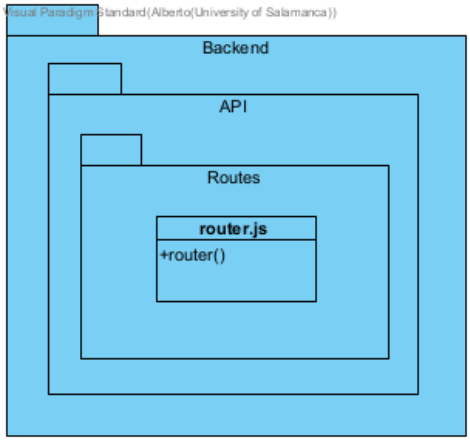

Figura 2.9: Router

2.3.2.2. Middlewares

Contiene las controladores del subsistema API encargados de comprobar ciertas condiciones.

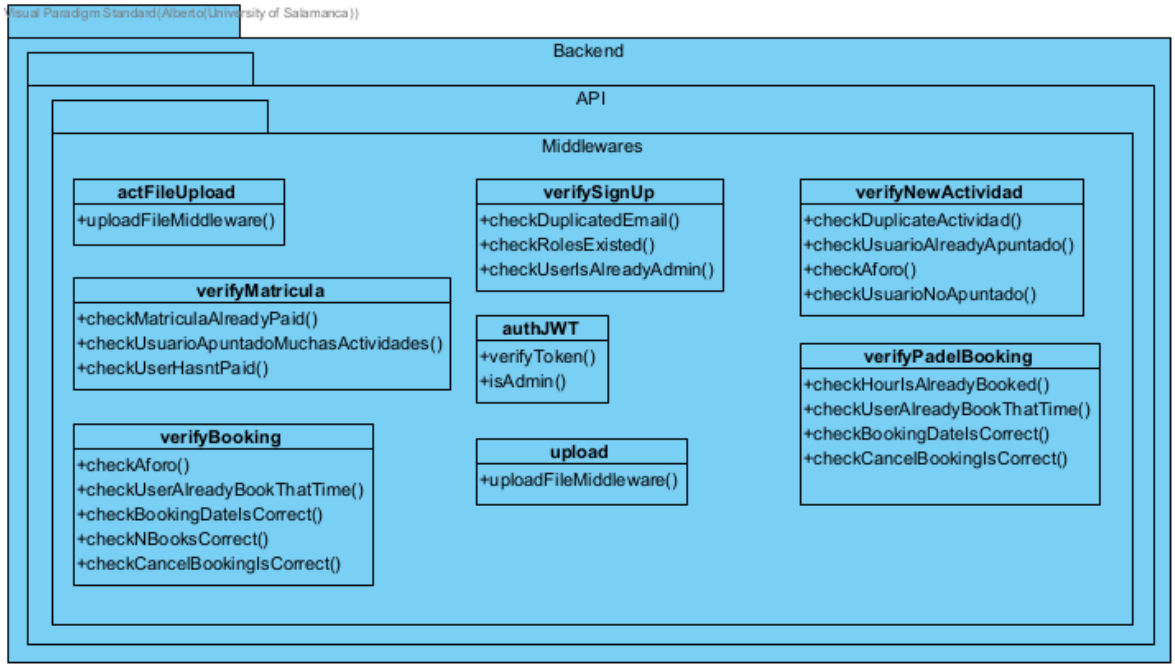

Figura 2.10: Middlewares

2.3.2.3. Controllers

Contiene las controladores del subsistema API encargados de gestionar las peticiones.

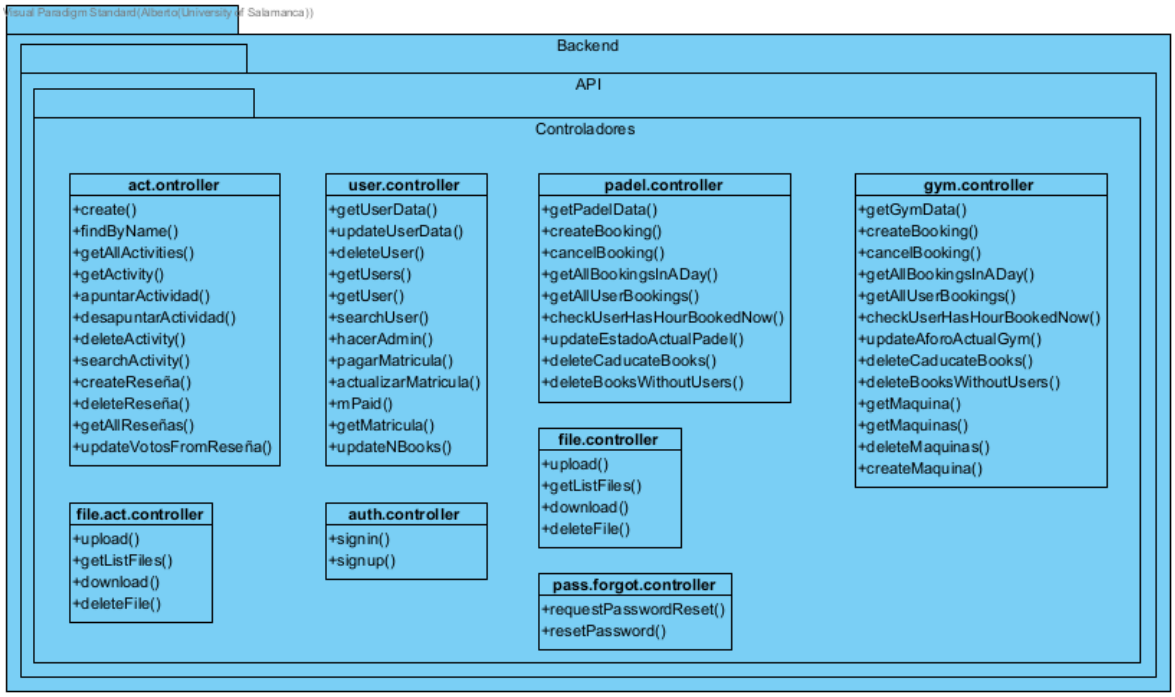

Figura 2.11: Controllers

2.3.2.4. Models

Contiene las entidades del sistema utilizadas por el subsistema API. Dichas entidades quedan guardadas en el sistema de manera persistente en una base de datos de MongoDB.

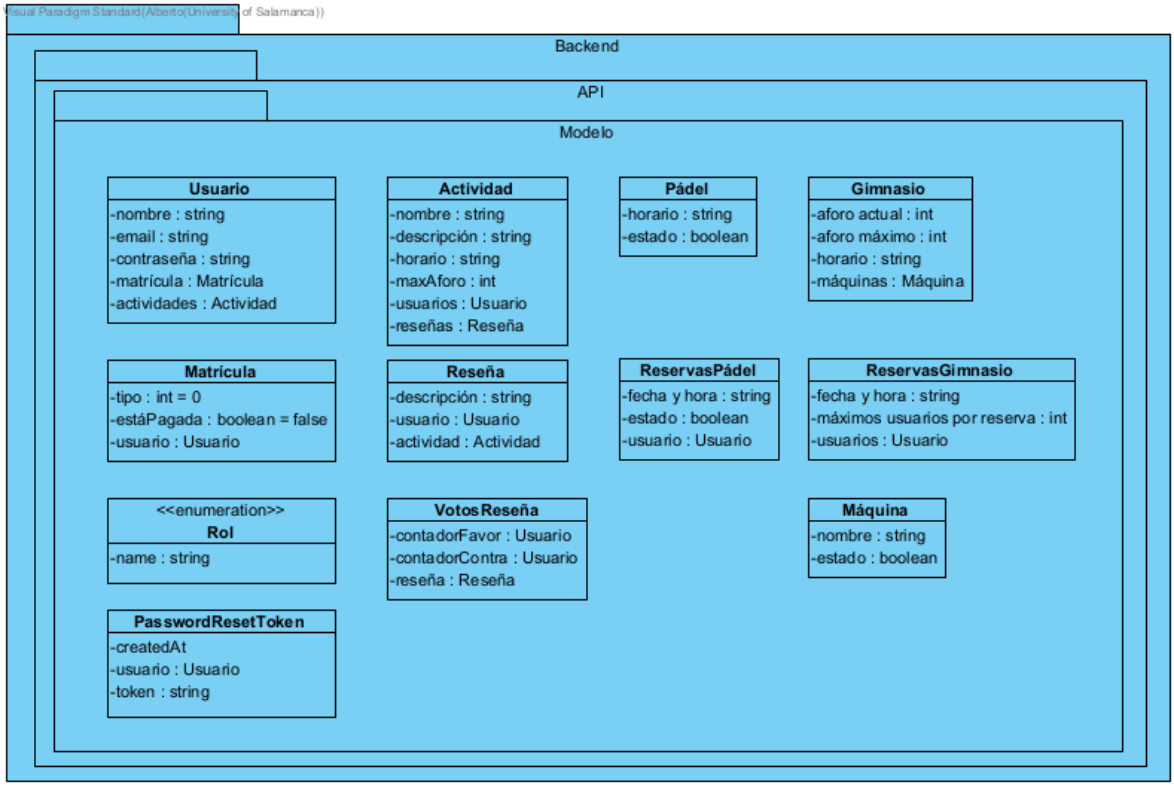

Figura 2.12: Models

2.3.2.5. Server

Se encarga de comenzar a escuchar peticiones y de iniciar la conexión con la base de datos.

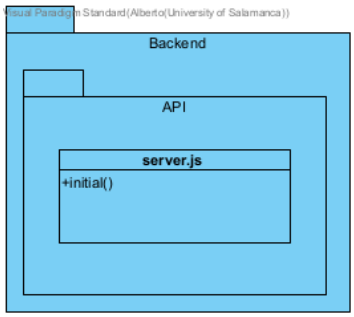

Figura 2.13: Server

2.3.3. Base de datos

Colecciones de la base de datos de MongoDB y sus relaciones.

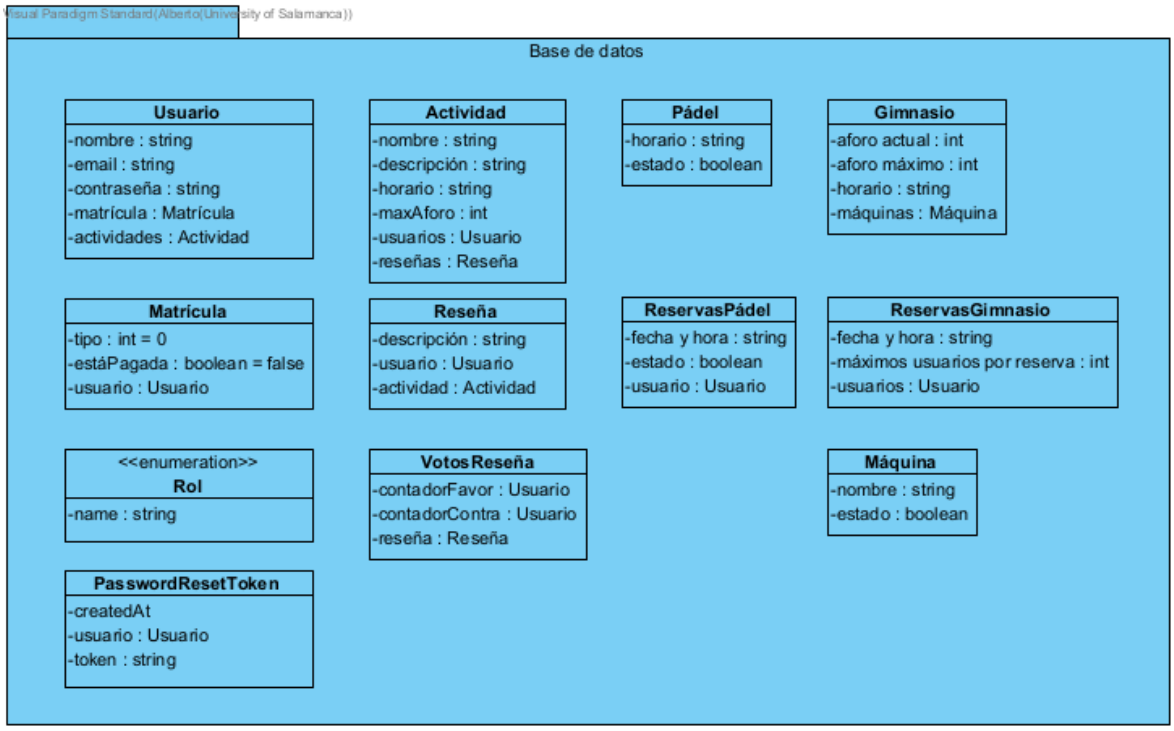

Figura 2.14: Base de datos

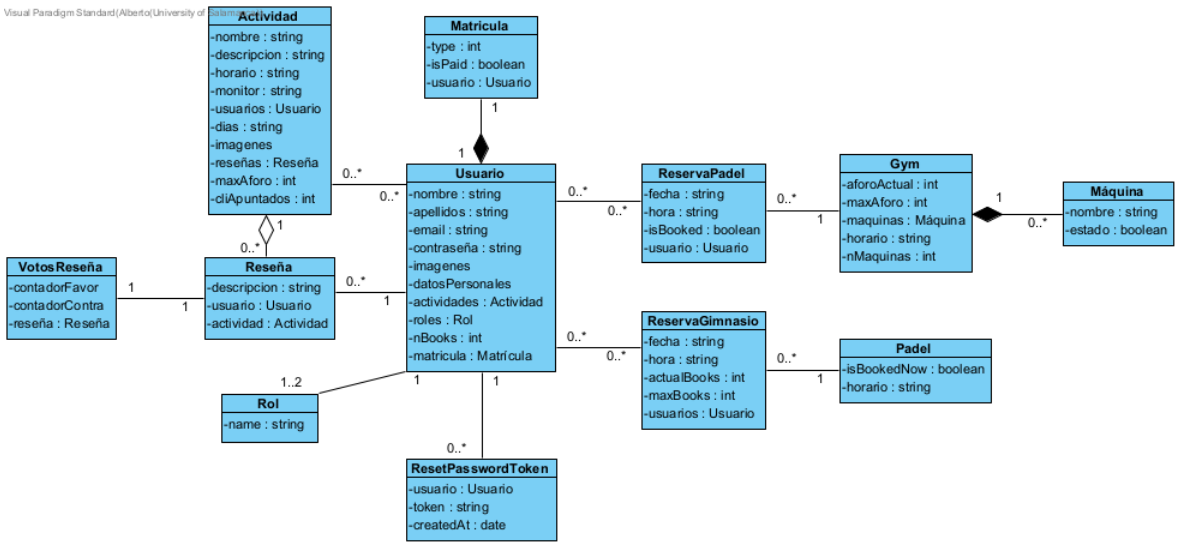

Figura 2.15: Diagrama de Clases de las relaciones de la base de datos

2.4. Vista arquitectónica

El objetivo de este apartado es otorgar de forma más detallada el patrón MVVM adaptado logrando así poder determinar de mejor forma las relaciones y divisiones del sistema.

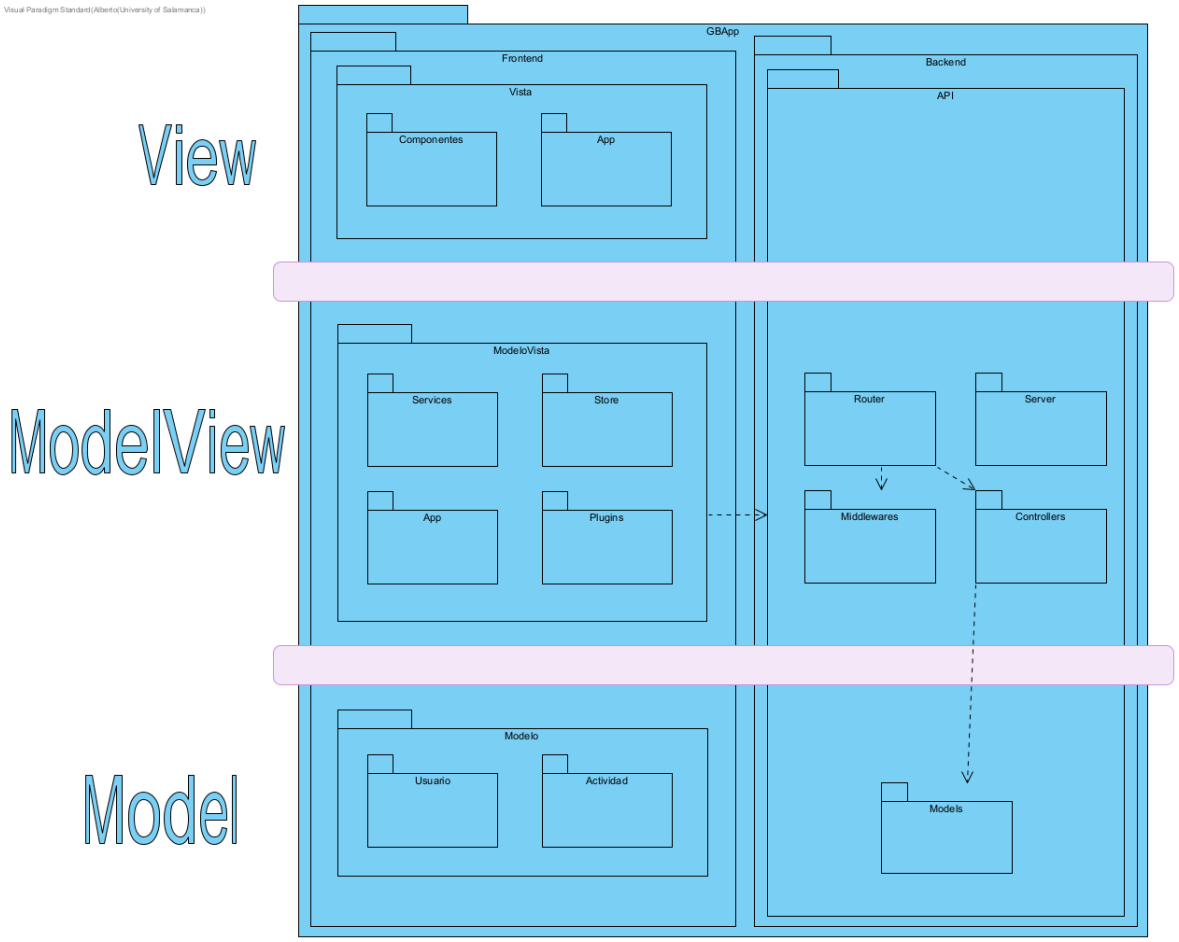

Figura 2.16: Vista arquitectónica del sistema

2.5. Realización de casos de uso

En este apartado se detallarán los intercambios de mensajes entre los objetos haciendo uso de diagramas de secuencia.

2.5.1. Gestión de autentificación

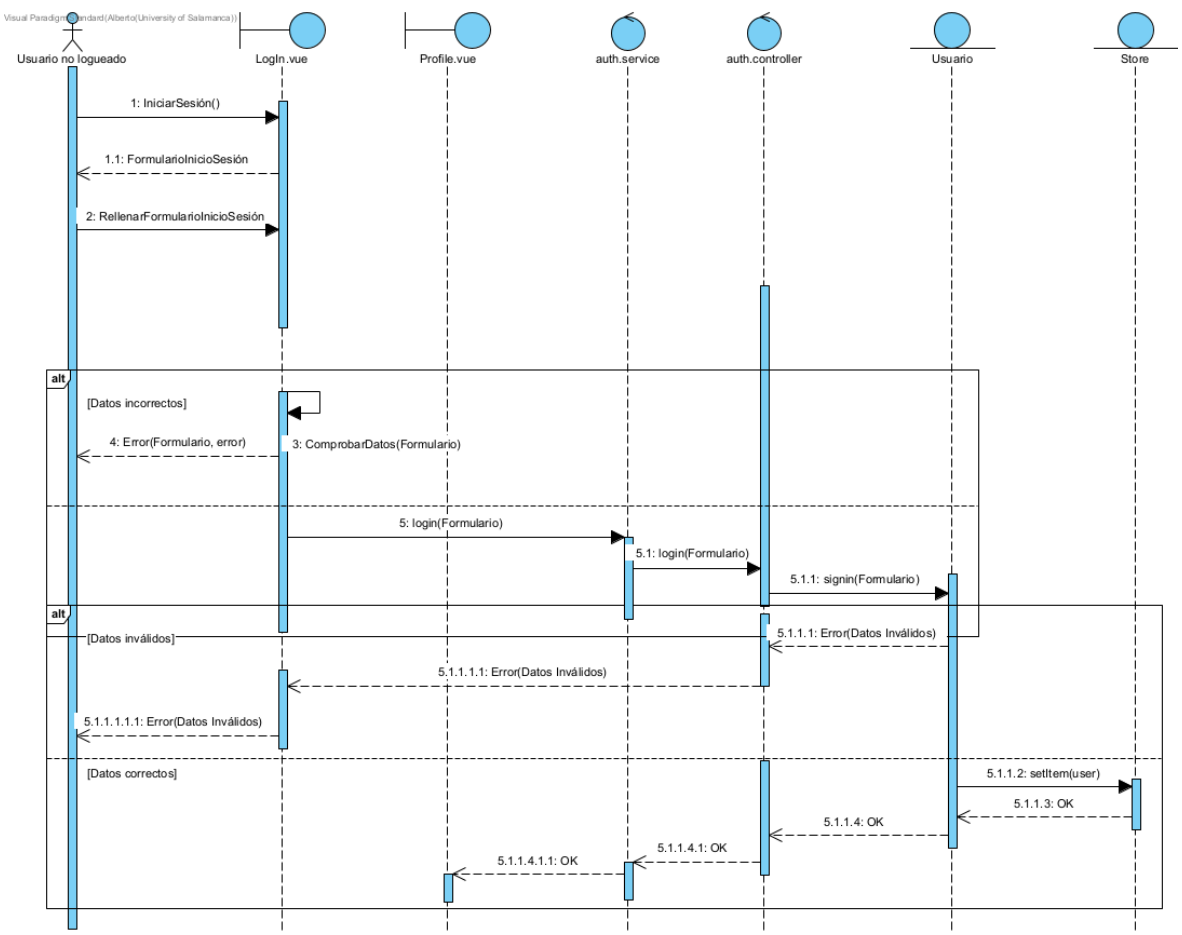

Figura 2.17: UC-0001 Iniciar sesión

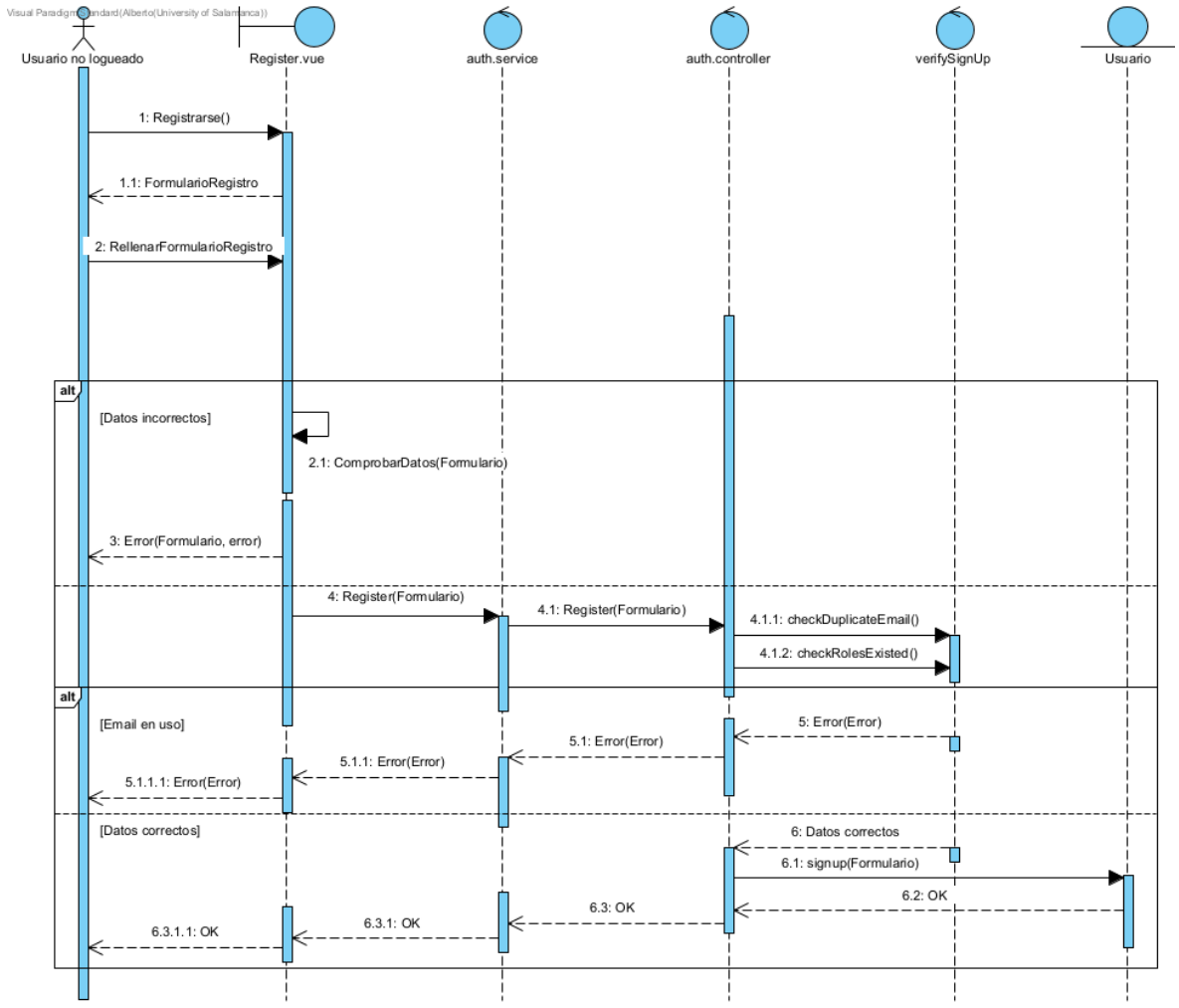

Figura 2.18: UC-0002 Registrarse

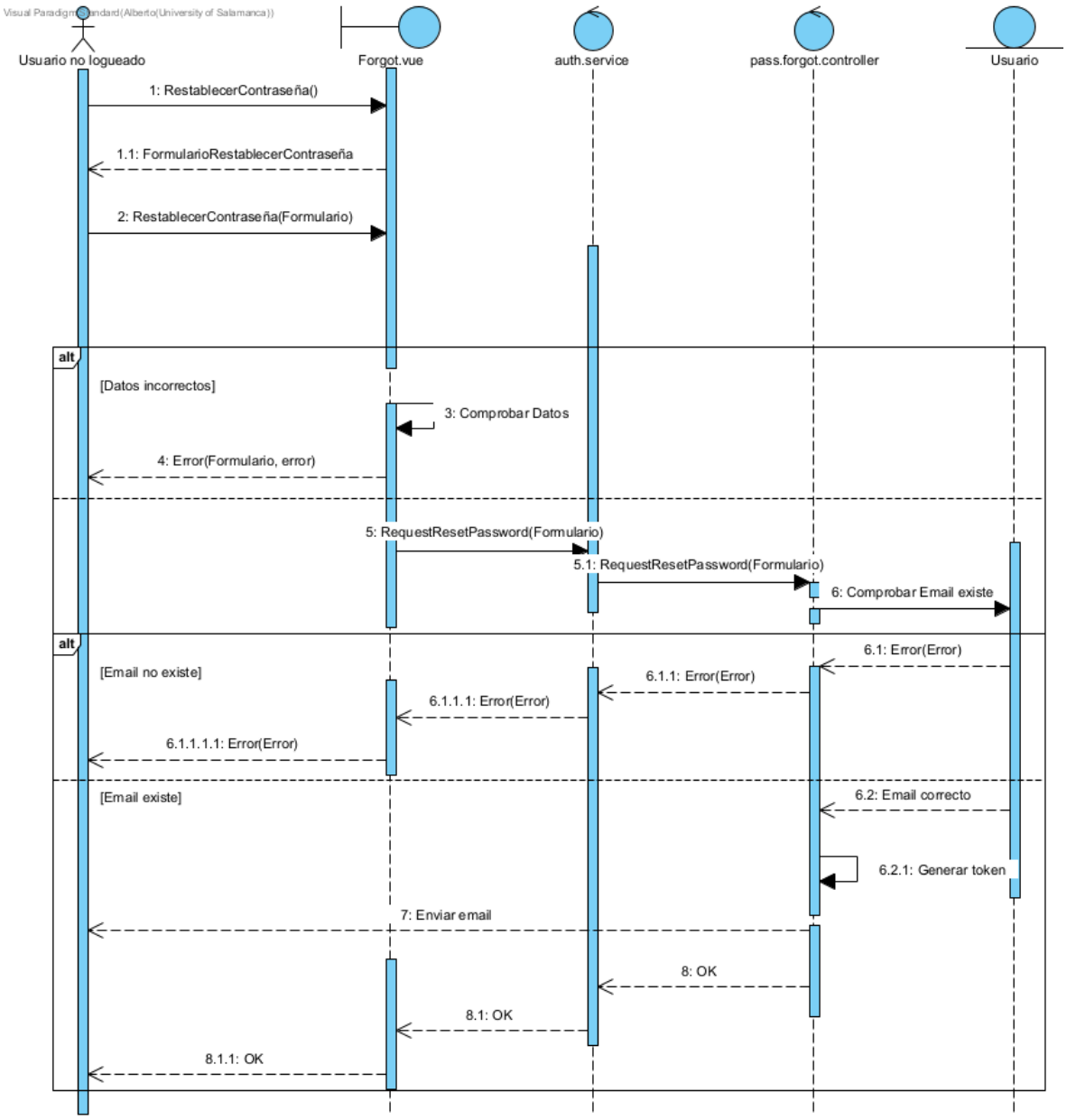

Figura 2.19: UC-0003 Solicitar restablecer contraseña

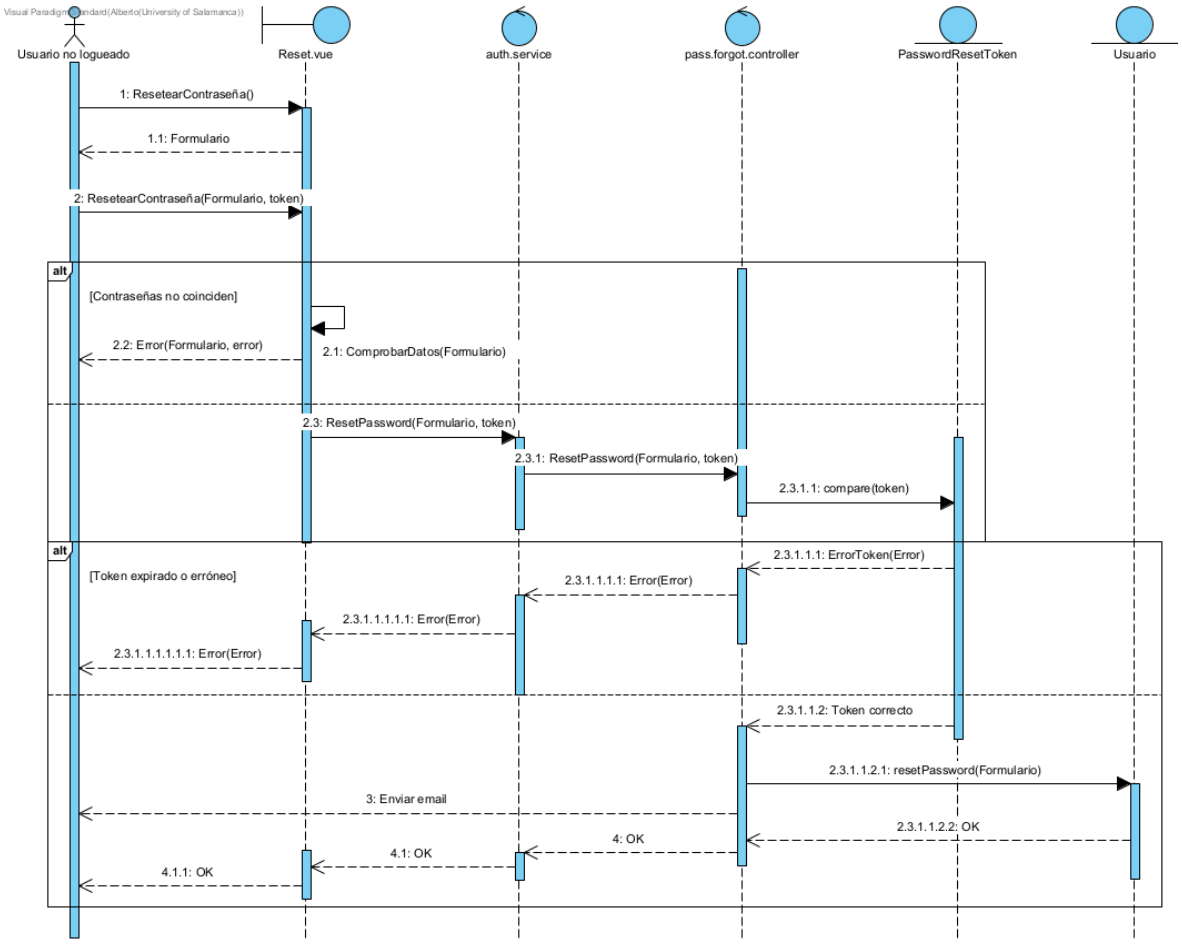

Figura 2.20: UC-0004 Resetear contraseña

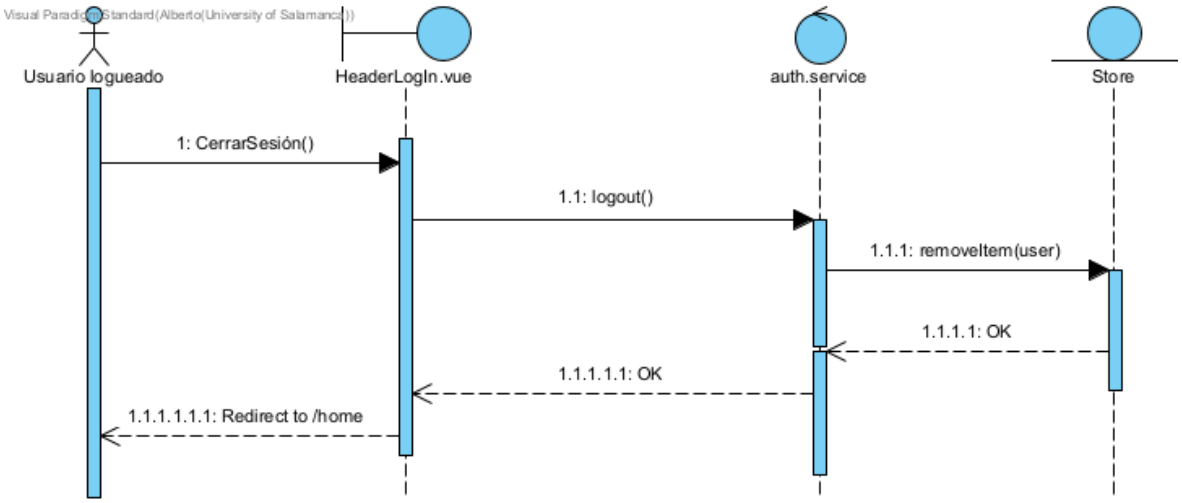

Figura 2.21: UC-0005 Cerrar sesión

2.5.2. Gestión de matrículas

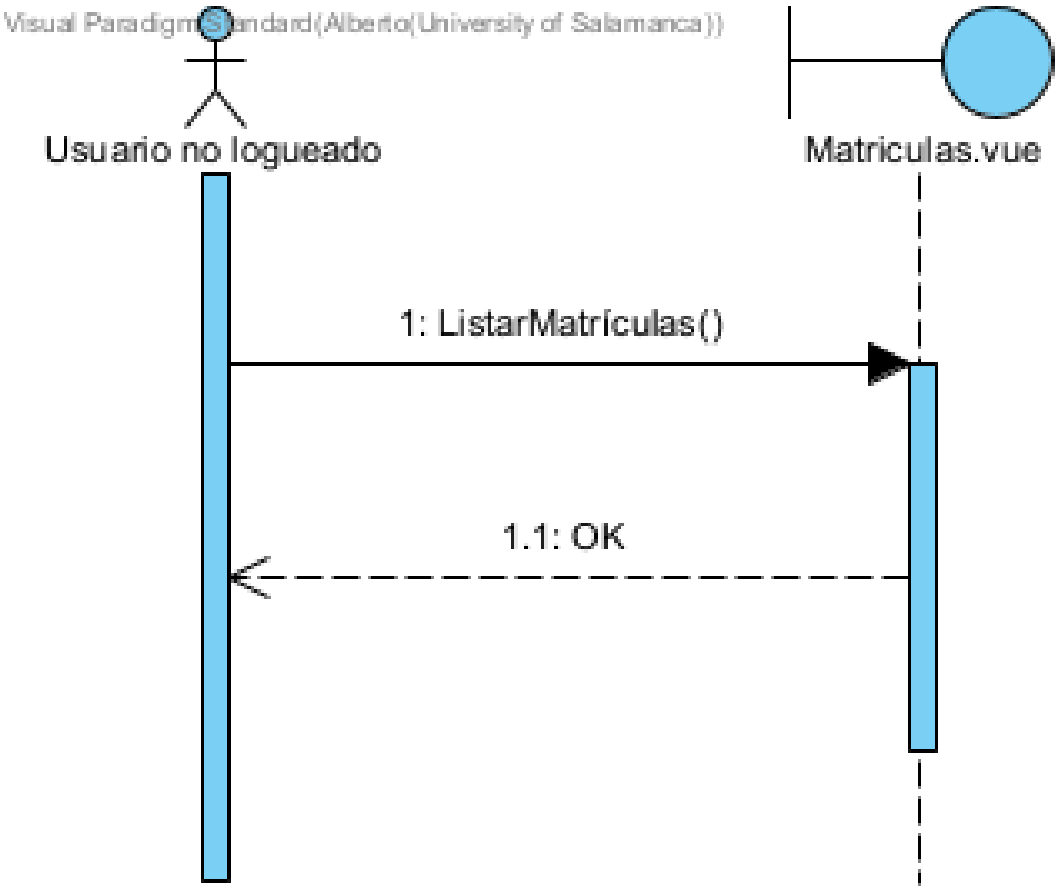

Figura 2.22: UC-0006 Listar matrículas

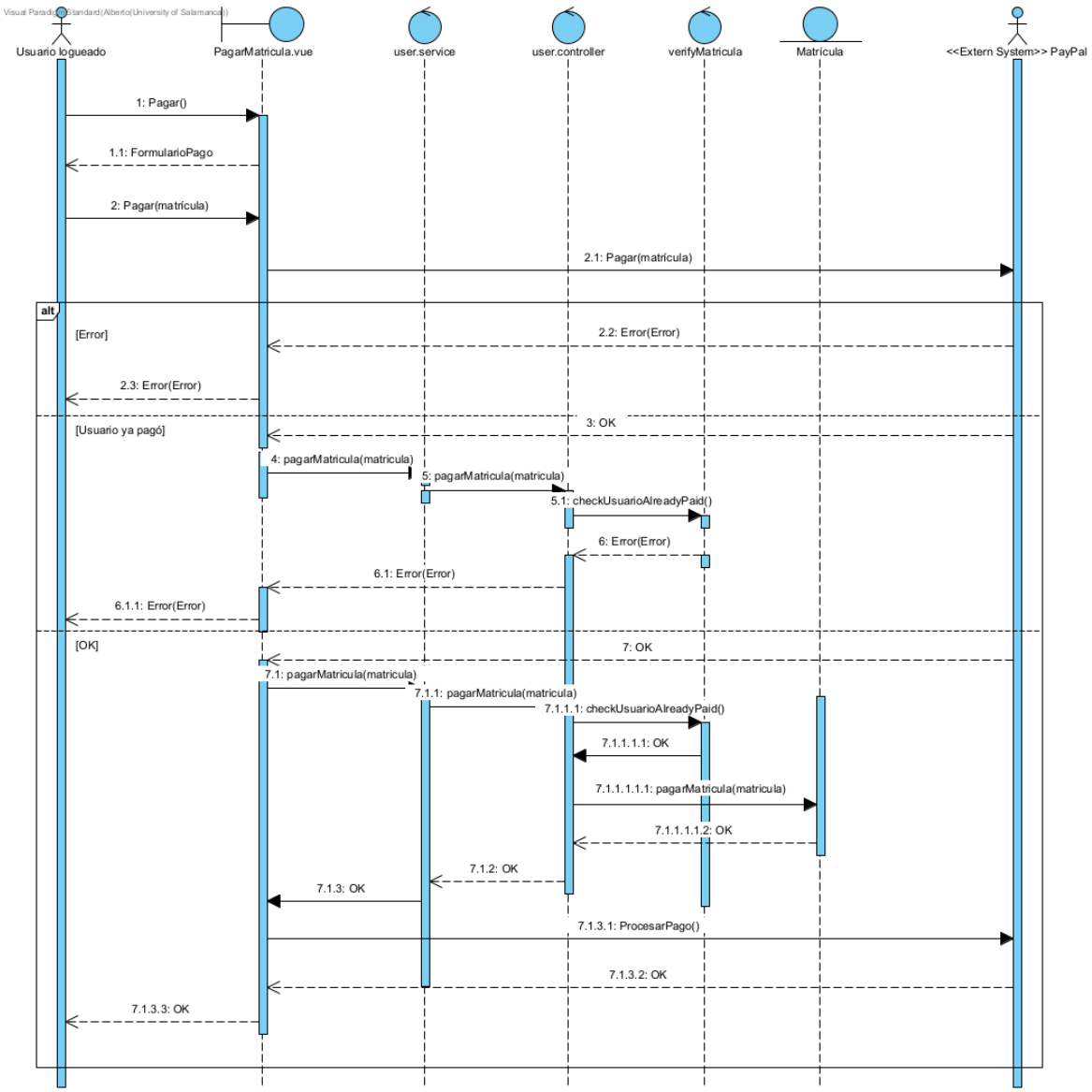

Figura 2.23: UC-0027 Pagar

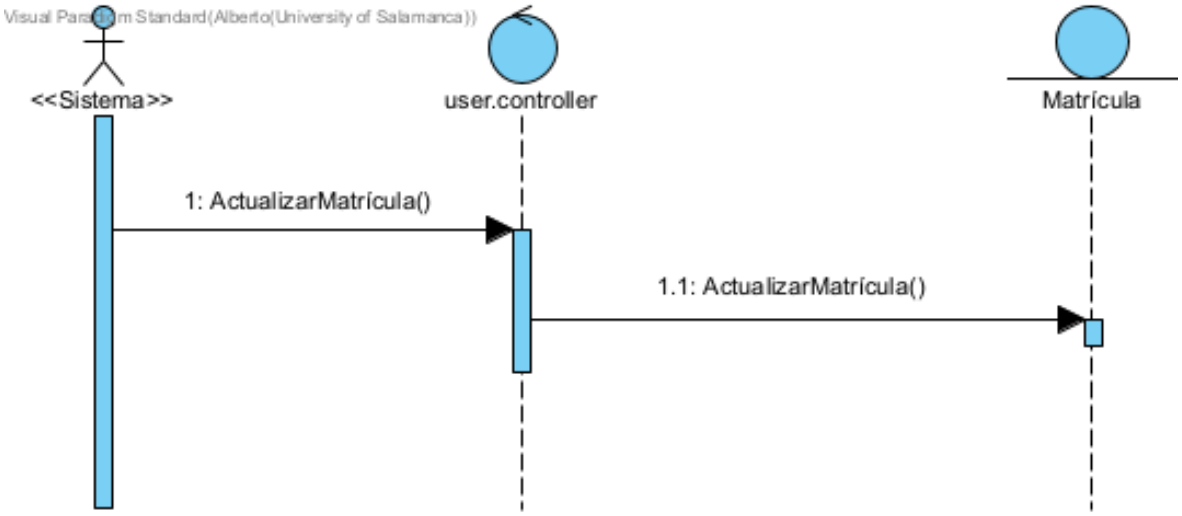

Figura 2.24: UC-0028 Actualizar matrículas

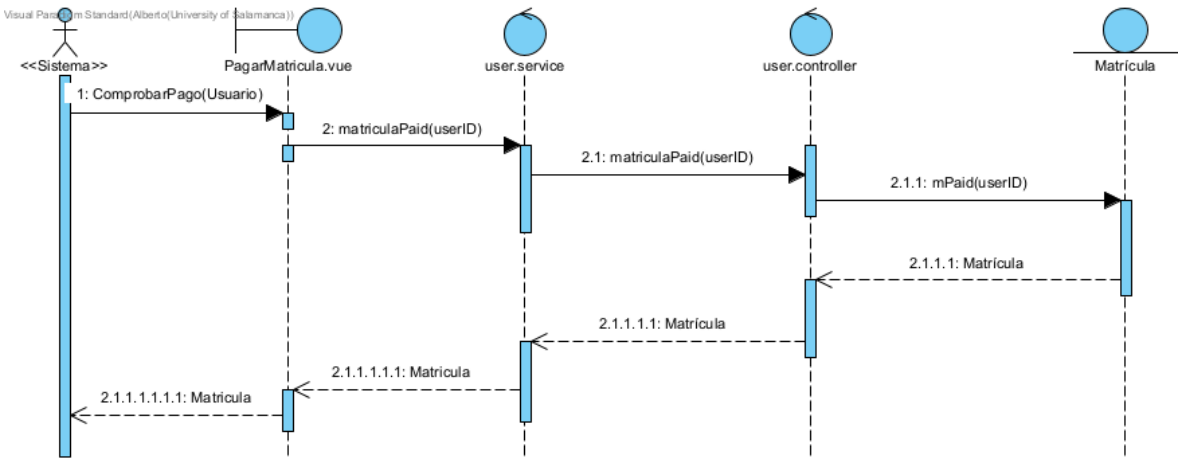

Figura 2.25: UC-0029 Comprobar pago

2.5.3. Gestión de usuario

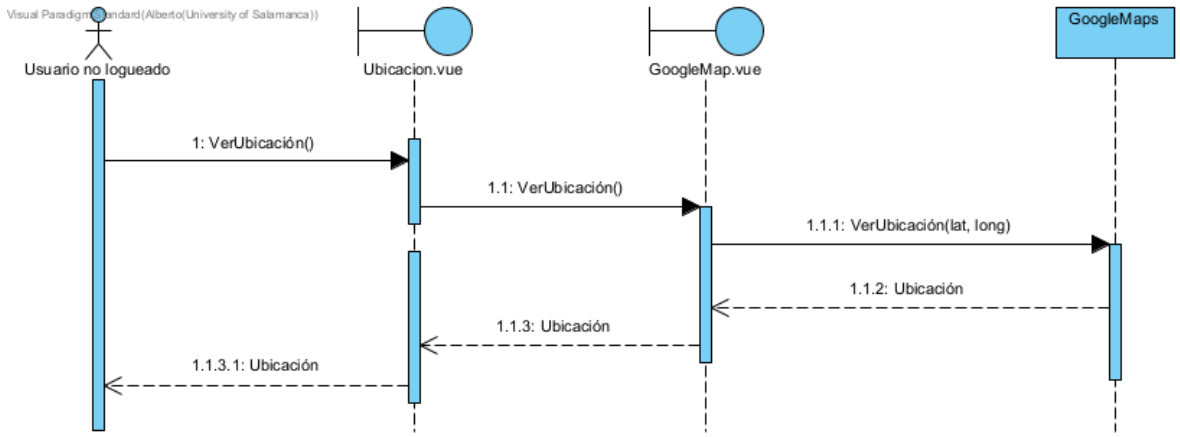

Figura 2.26: UC-0007 Ver ubicación

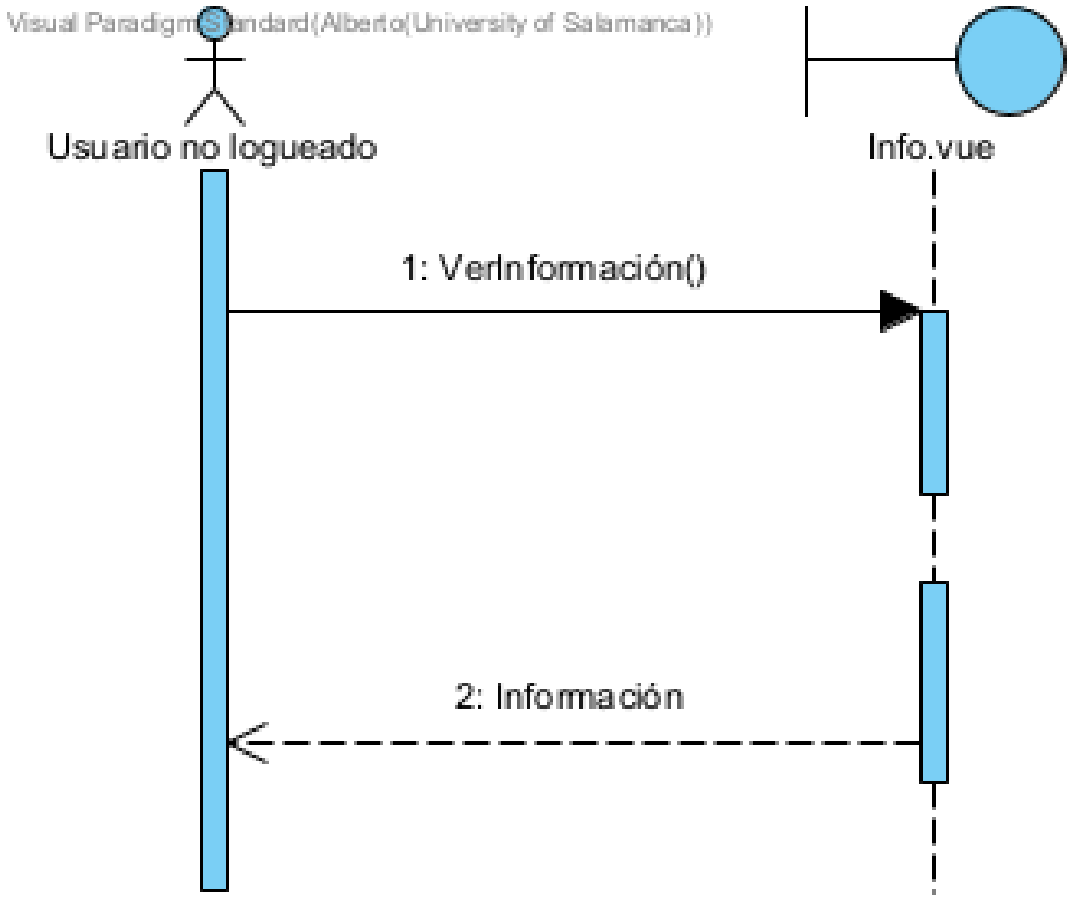

Figura 2.27: UC-0008 Ver información

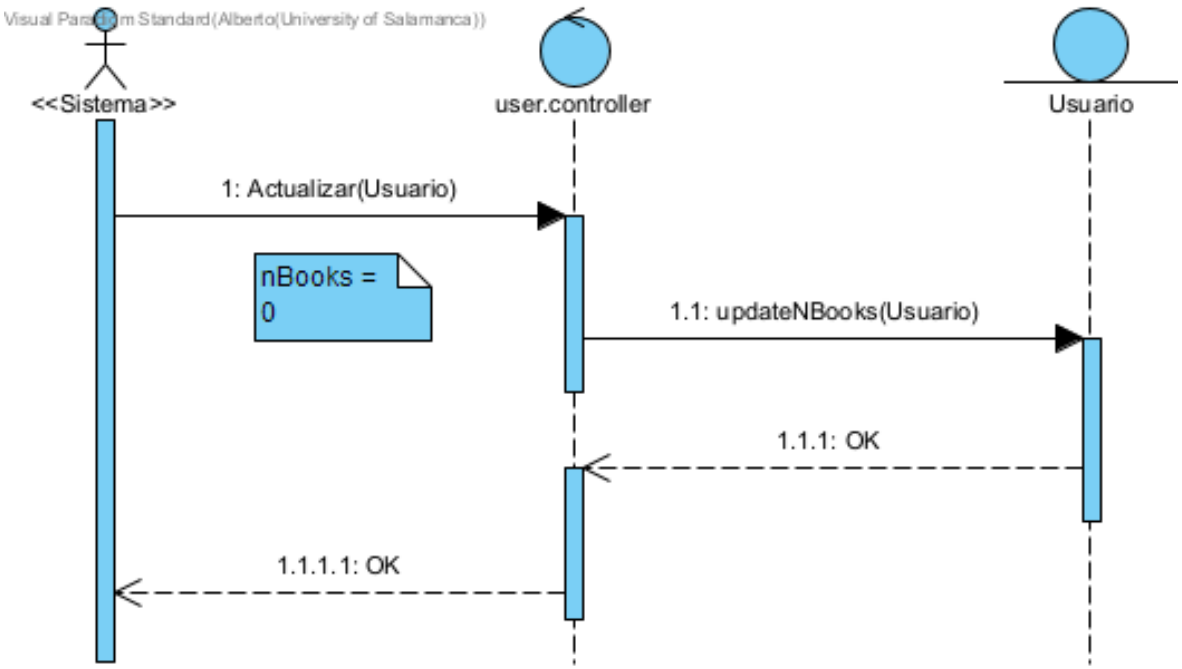

Figura 2.28: UC-0051 Actualizar Nºreservas semanales

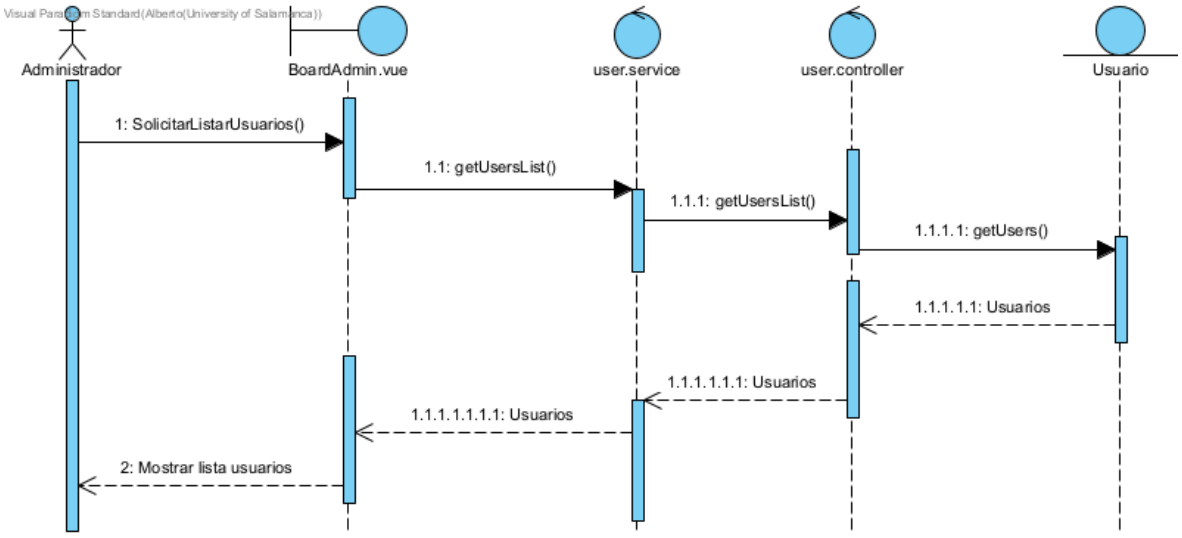

Figura 2.29: UC-0052 Listar usuarios

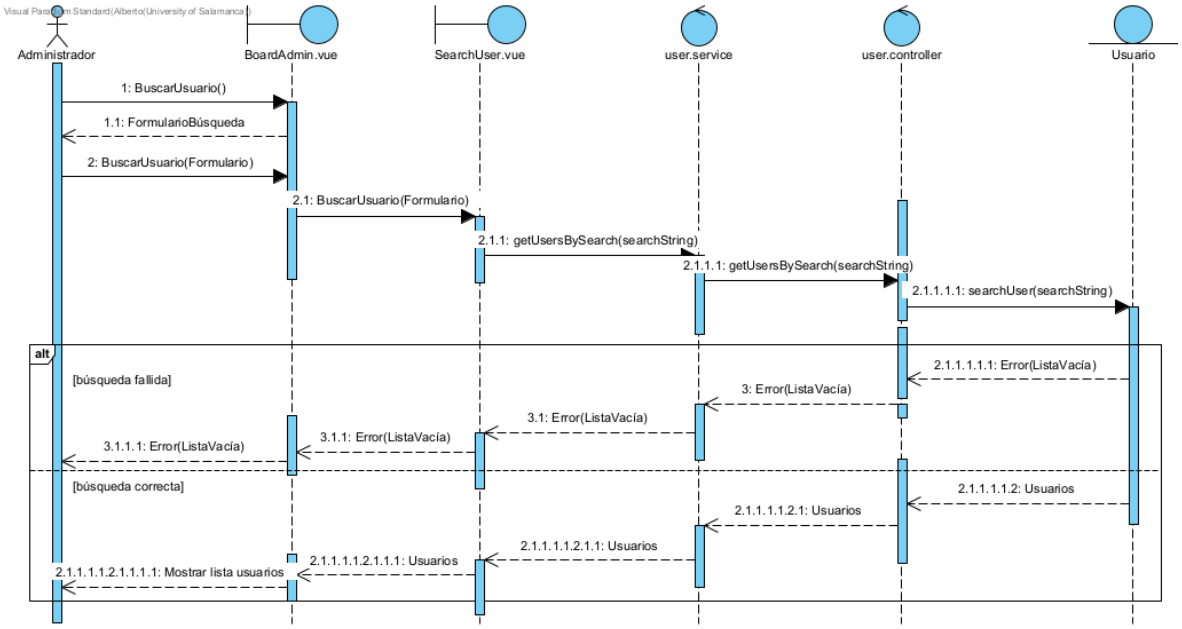

Figura 2.30: UC-0053 Buscar usuario

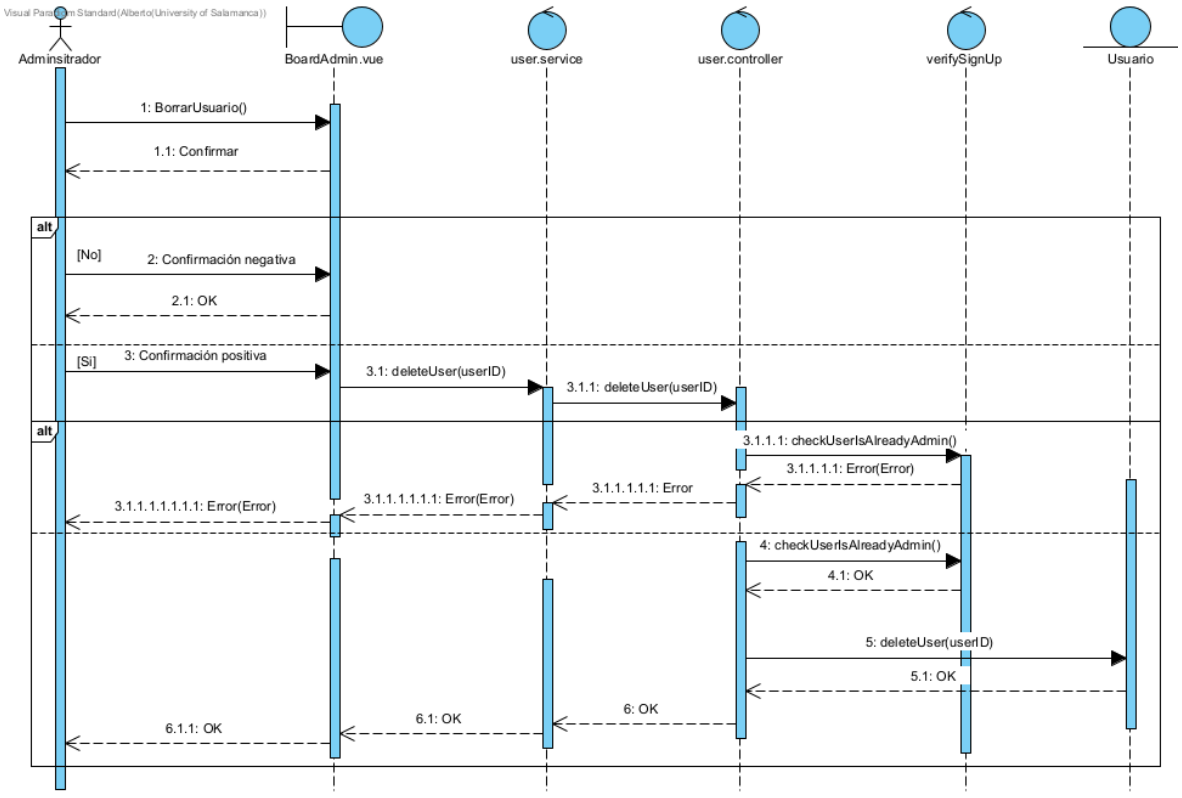

Figura 2.31: UC-0054 Borrar usuario

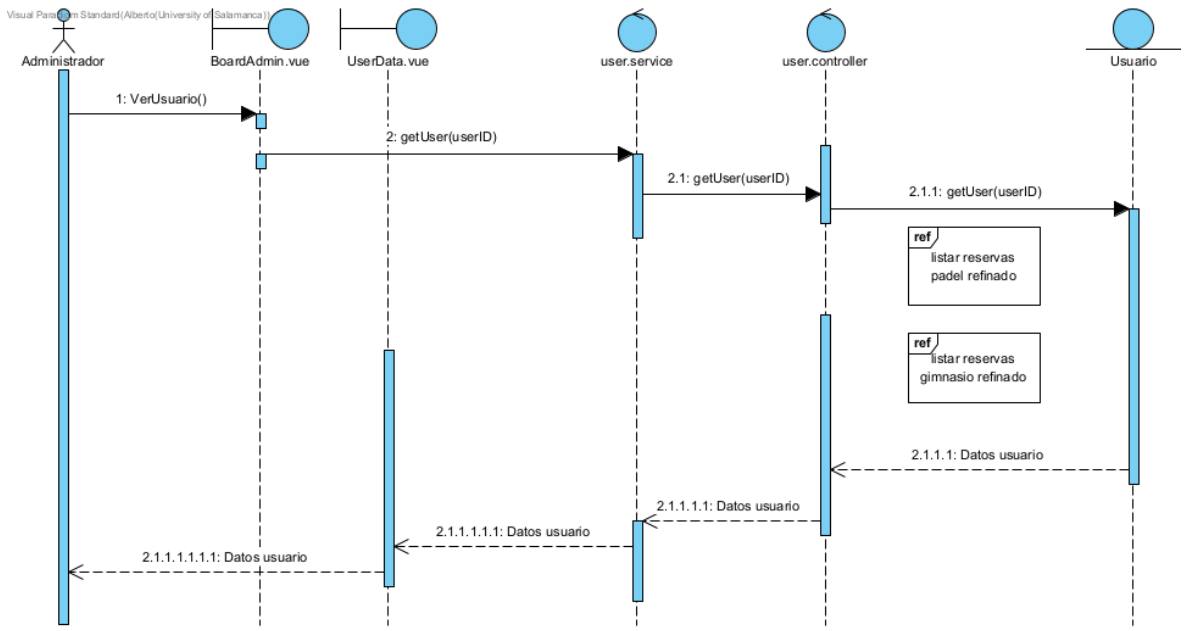

Figura 2.32: UC-0055 Ver usuario

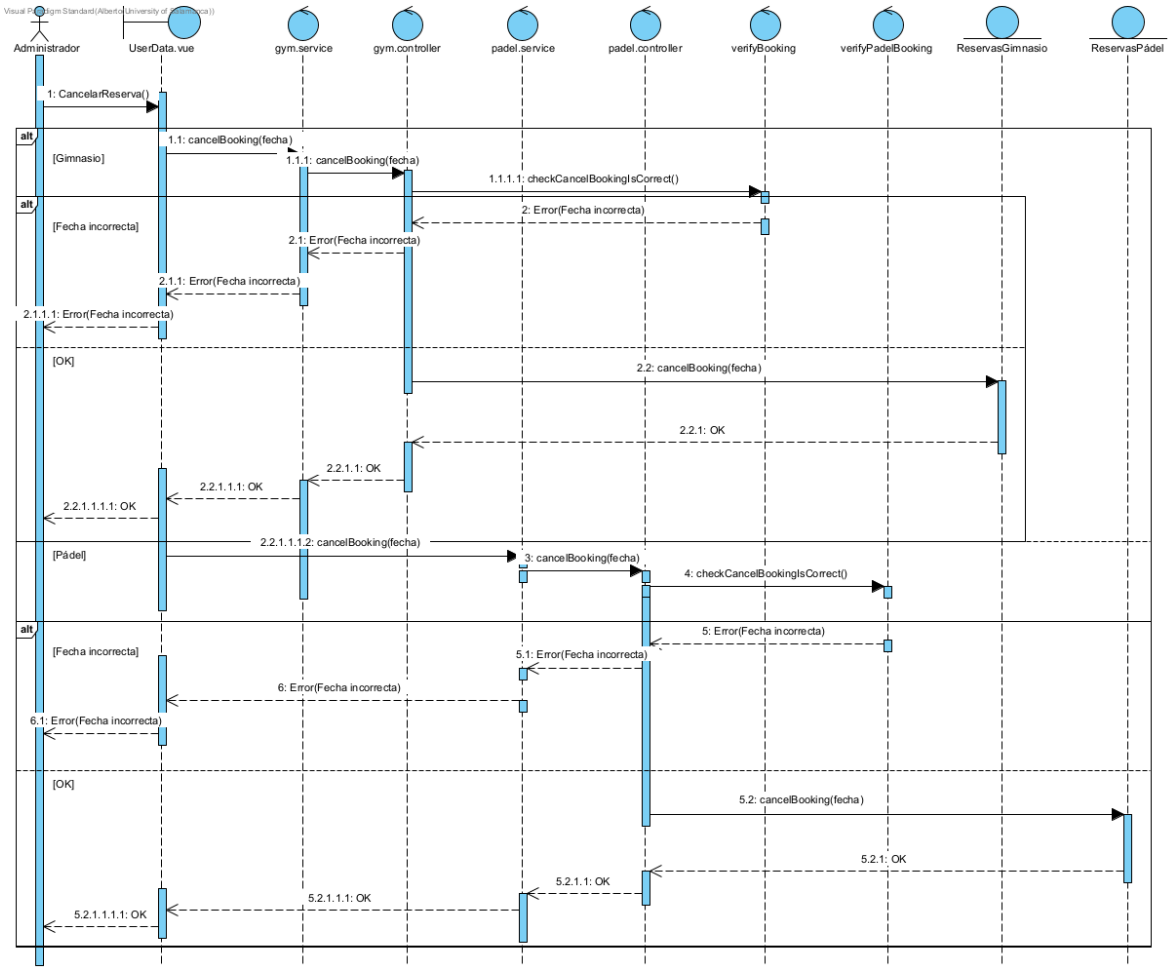

Figura 2.33: UC-0056 Cancelar reserva de usuario

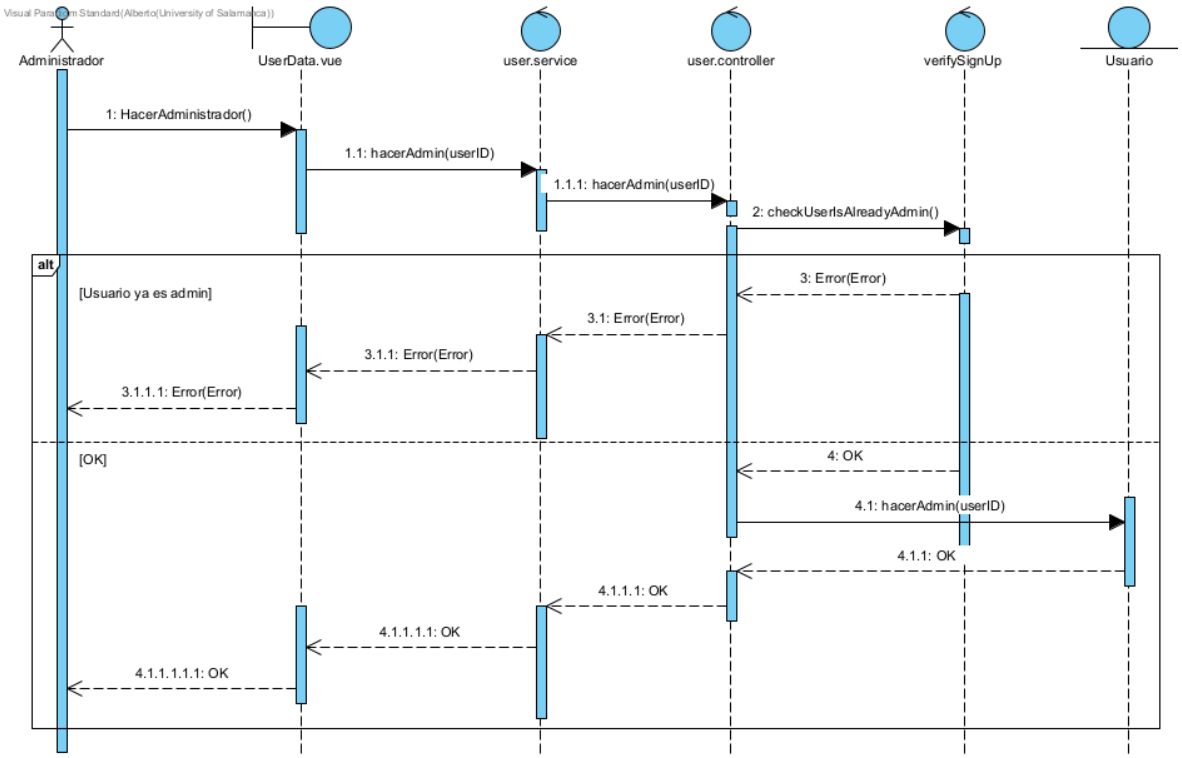

Figura 2.34: UC-0057 Hacer administrador

2.5.4. Gestión de actividades

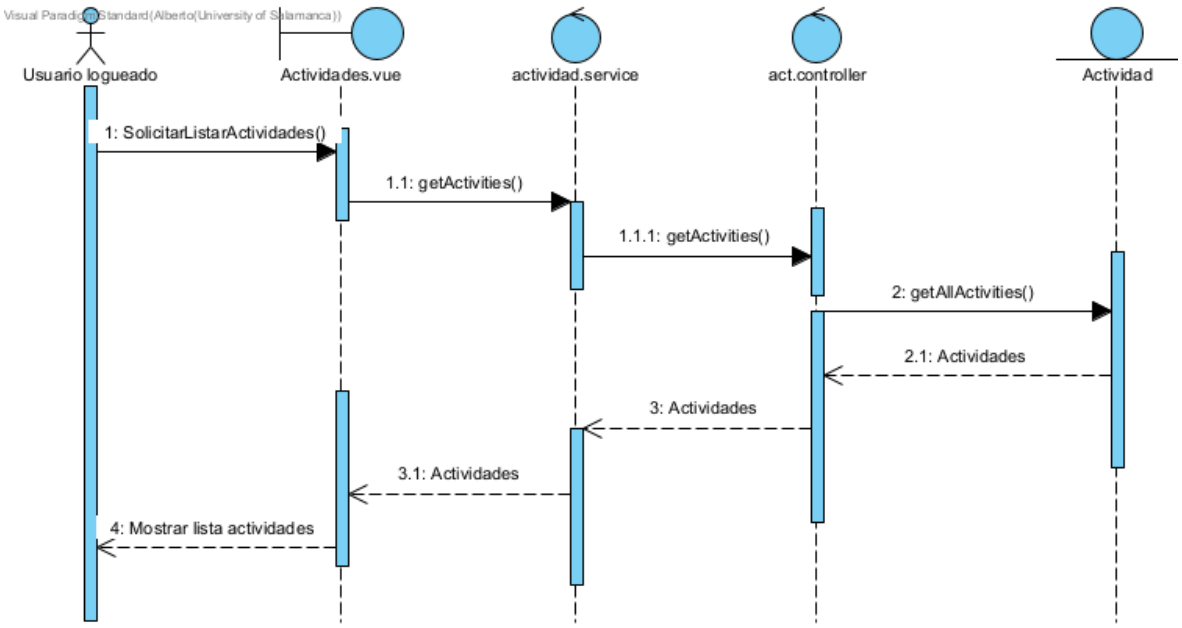

Figura 2.35: UC-0009 Listar actividades

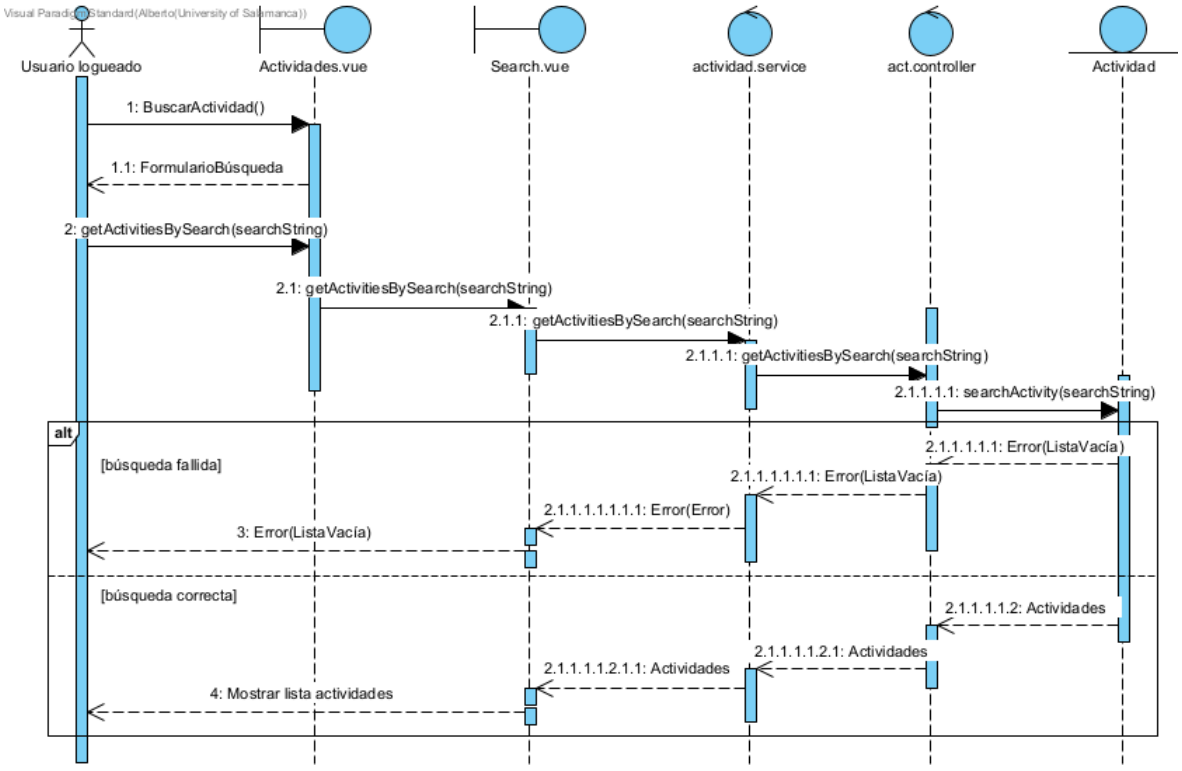

Figura 2.36: UC-0010 Buscar actividad

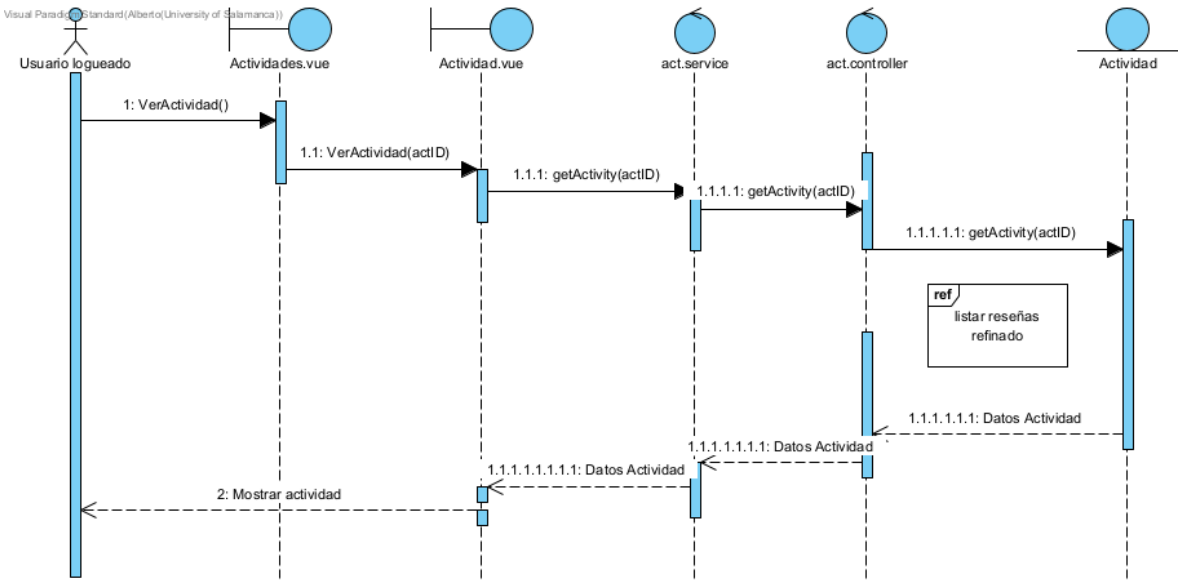

Figura 2.37: UC-0011 Ver actividad

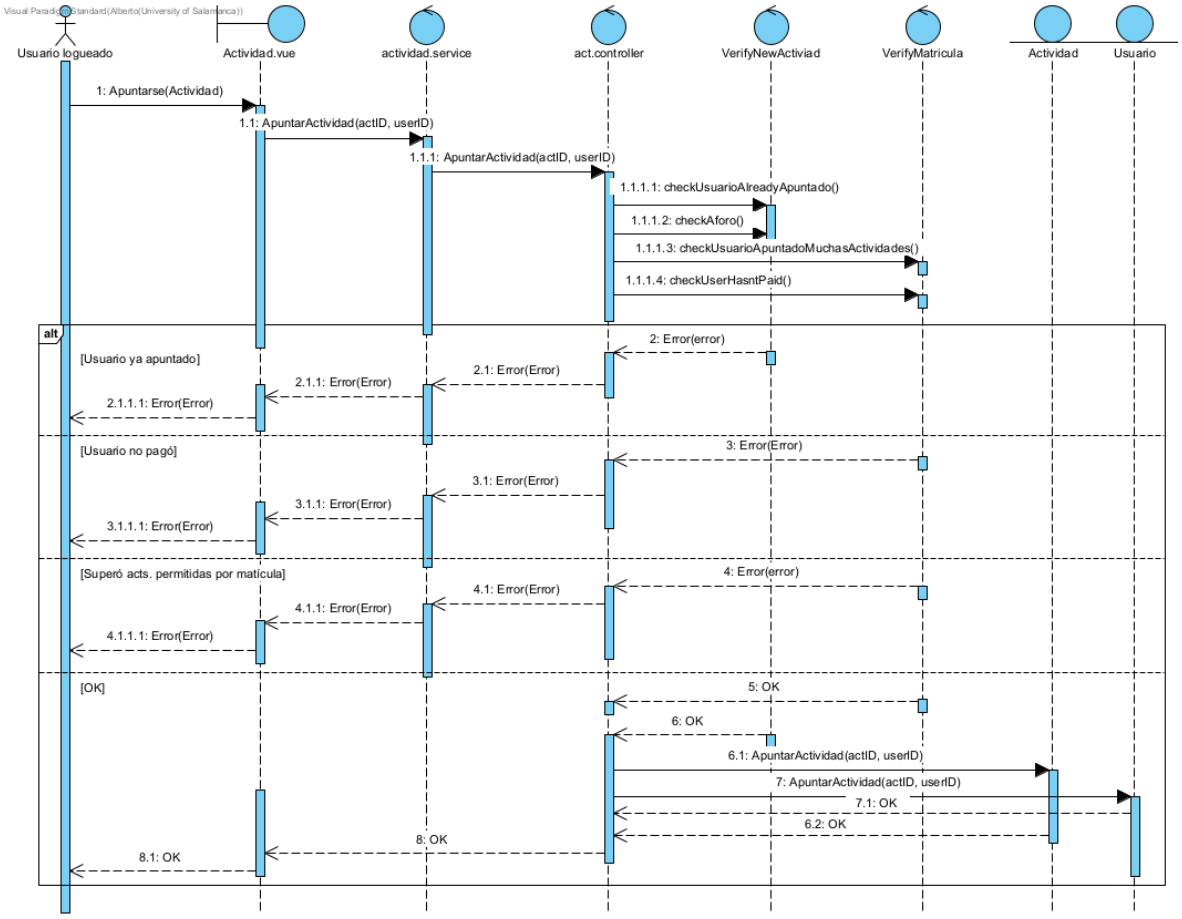

Figura 2.38: UC-0012 Apuntar actividad

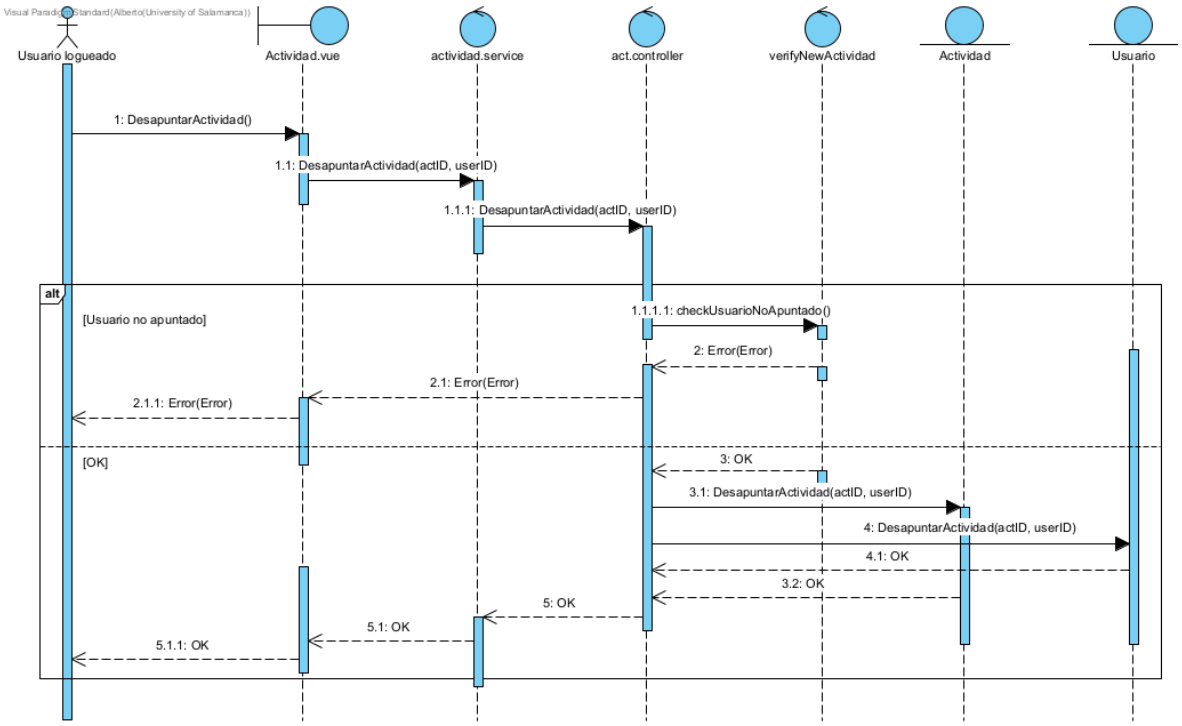

Figura 2.39: UC-0013 Desapuntar actividad

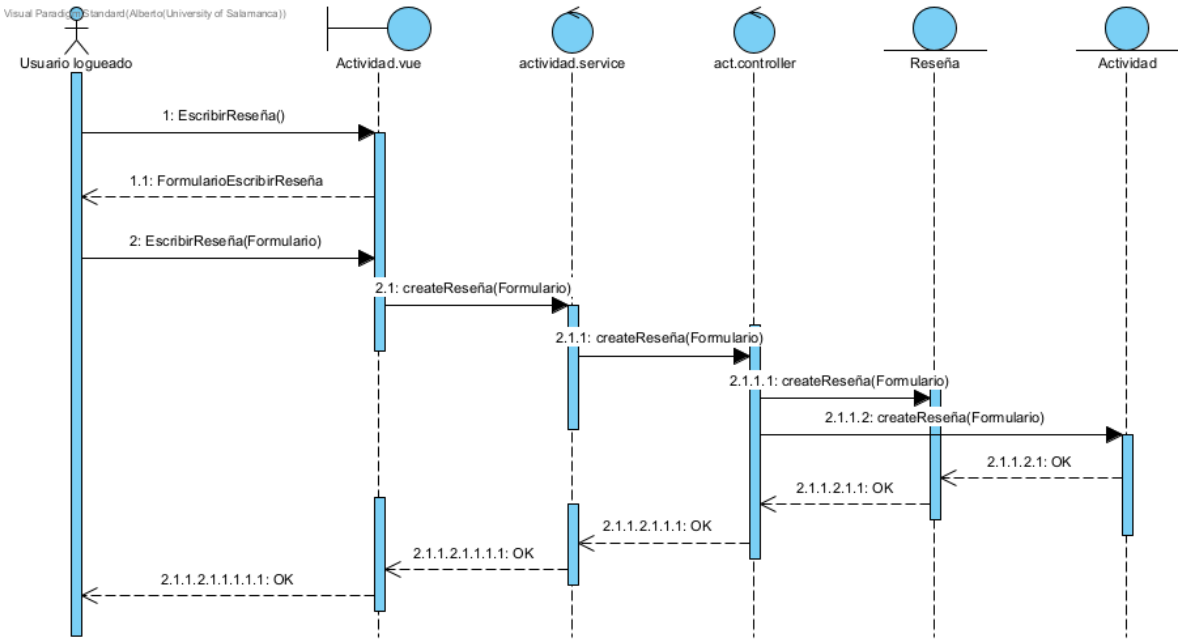

Figura 2.40: UC-0014 Escribir reseña

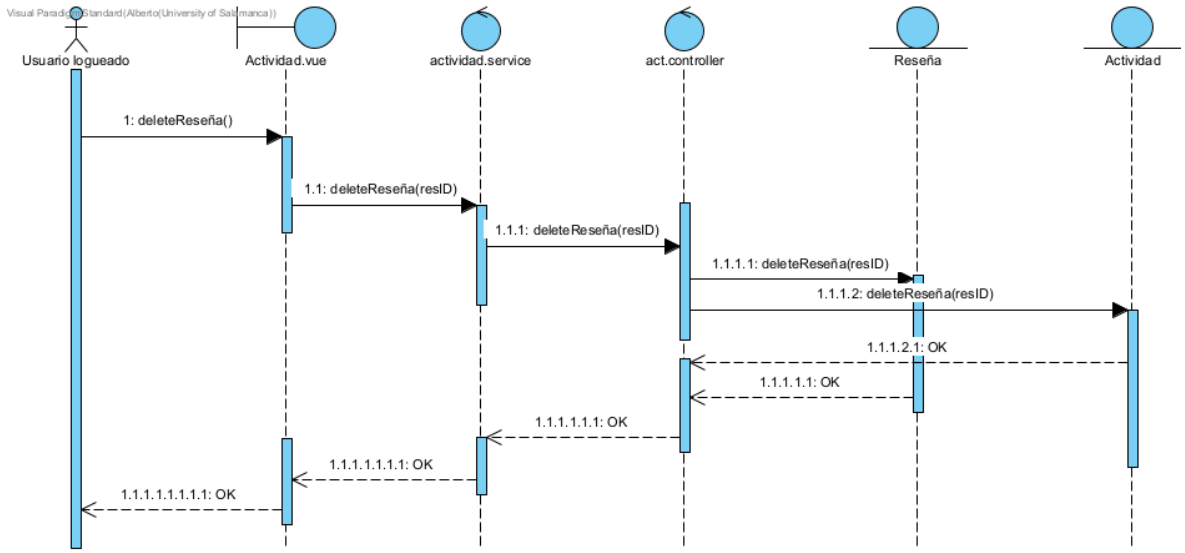

Figura 2.41: UC-0015 Borrar reseña

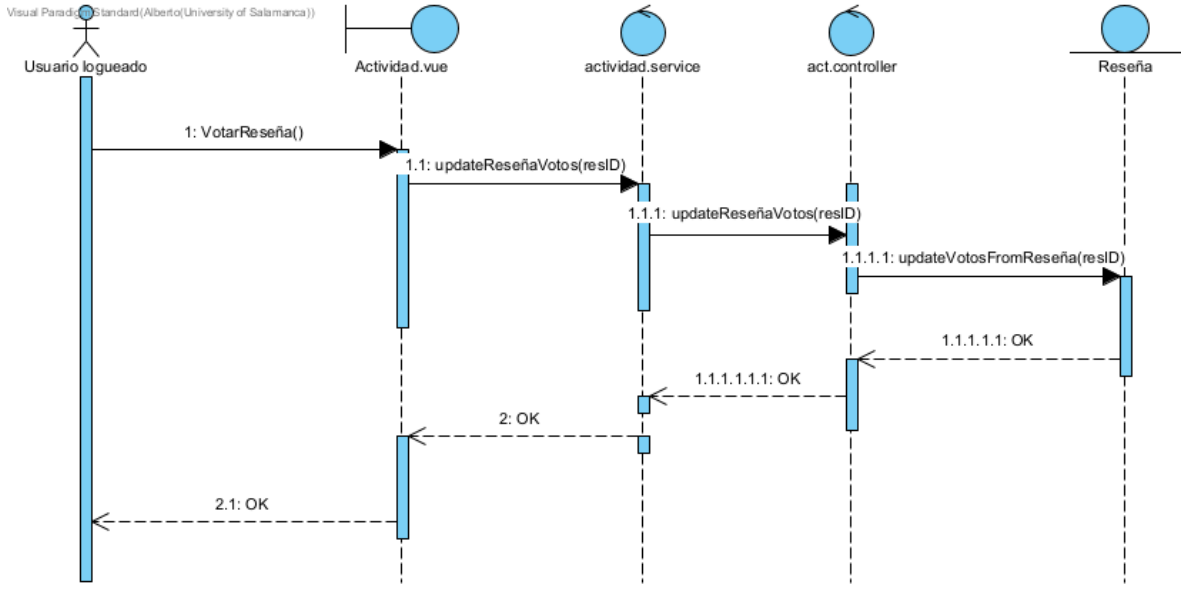

Figura 2.42: UC-0016 Votar reseña

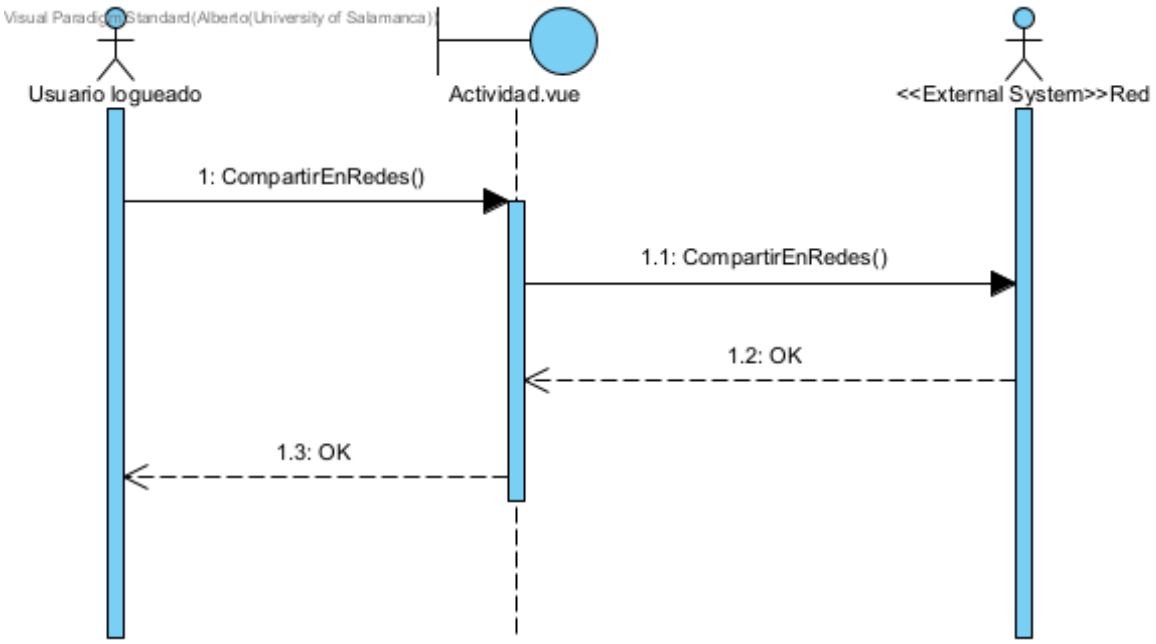

Figura 2.43: UC-0017 Compartir en las redes

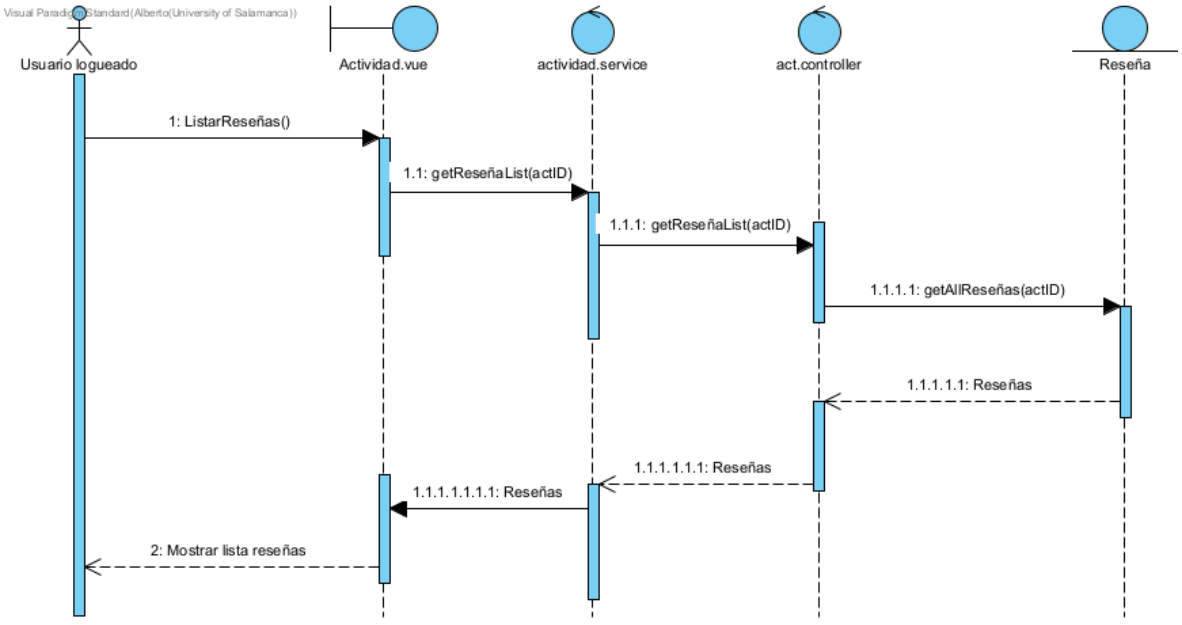

Figura 2.44: UC-0018 Listar reseñas

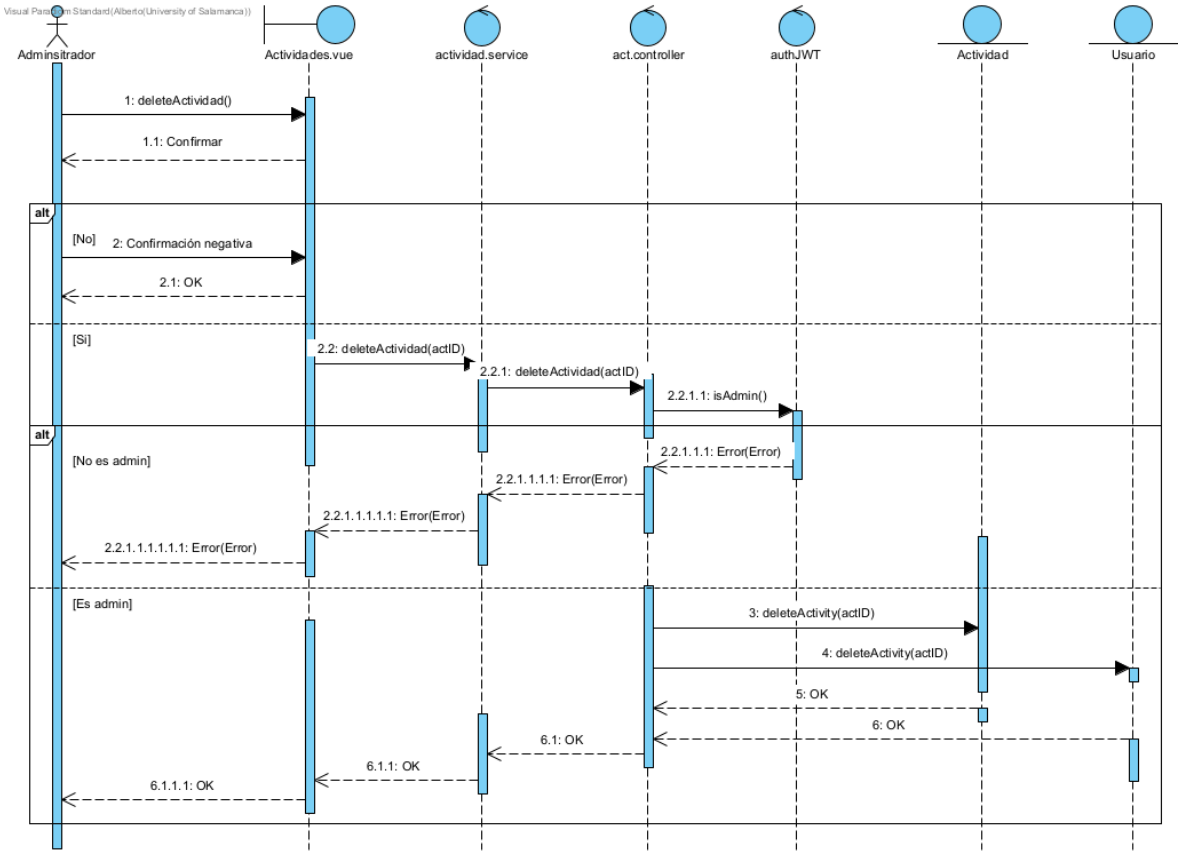

Figura 2.45: UC-0019 Borrar actividad

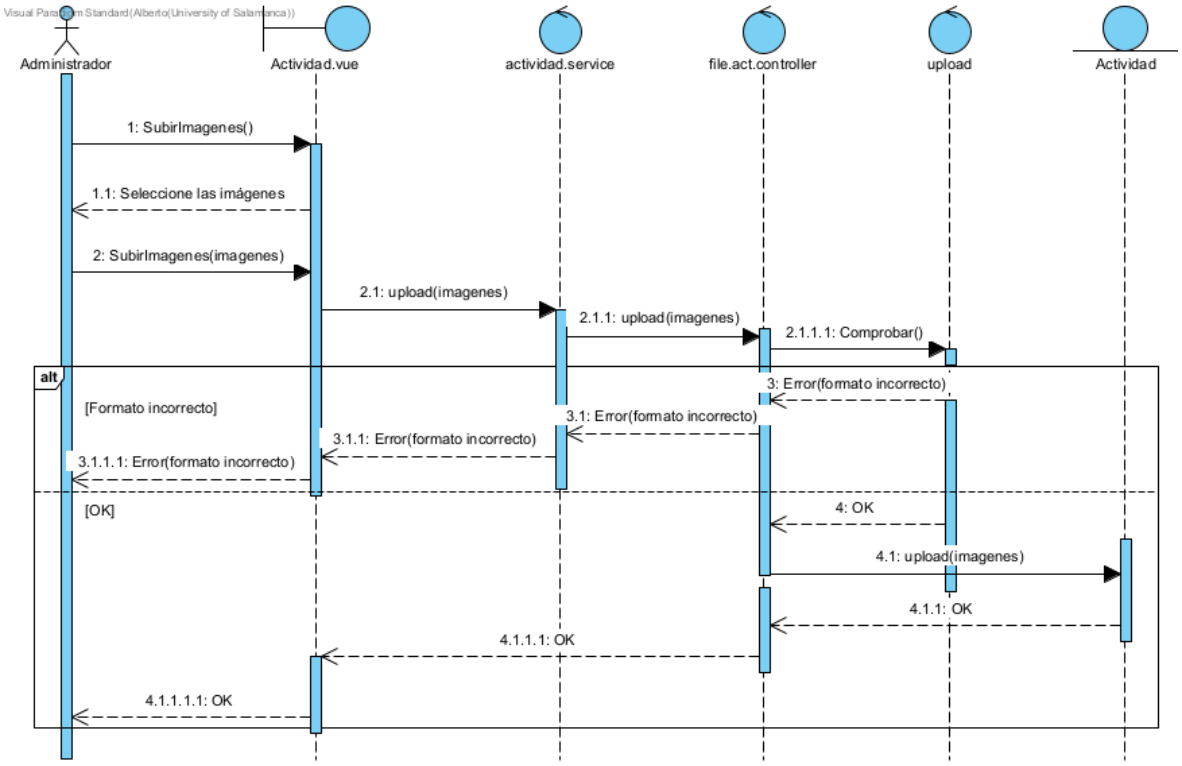

Figura 2.46: UC-0020 Subir imagen

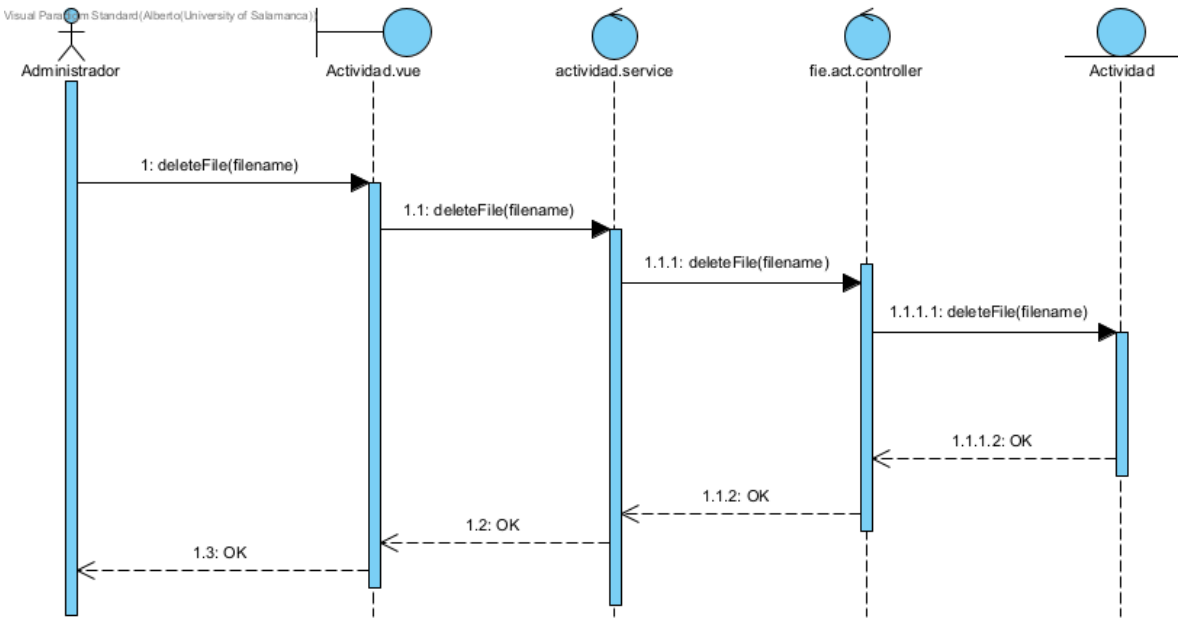

Figura 2.47: UC-0021 Borrar imagen

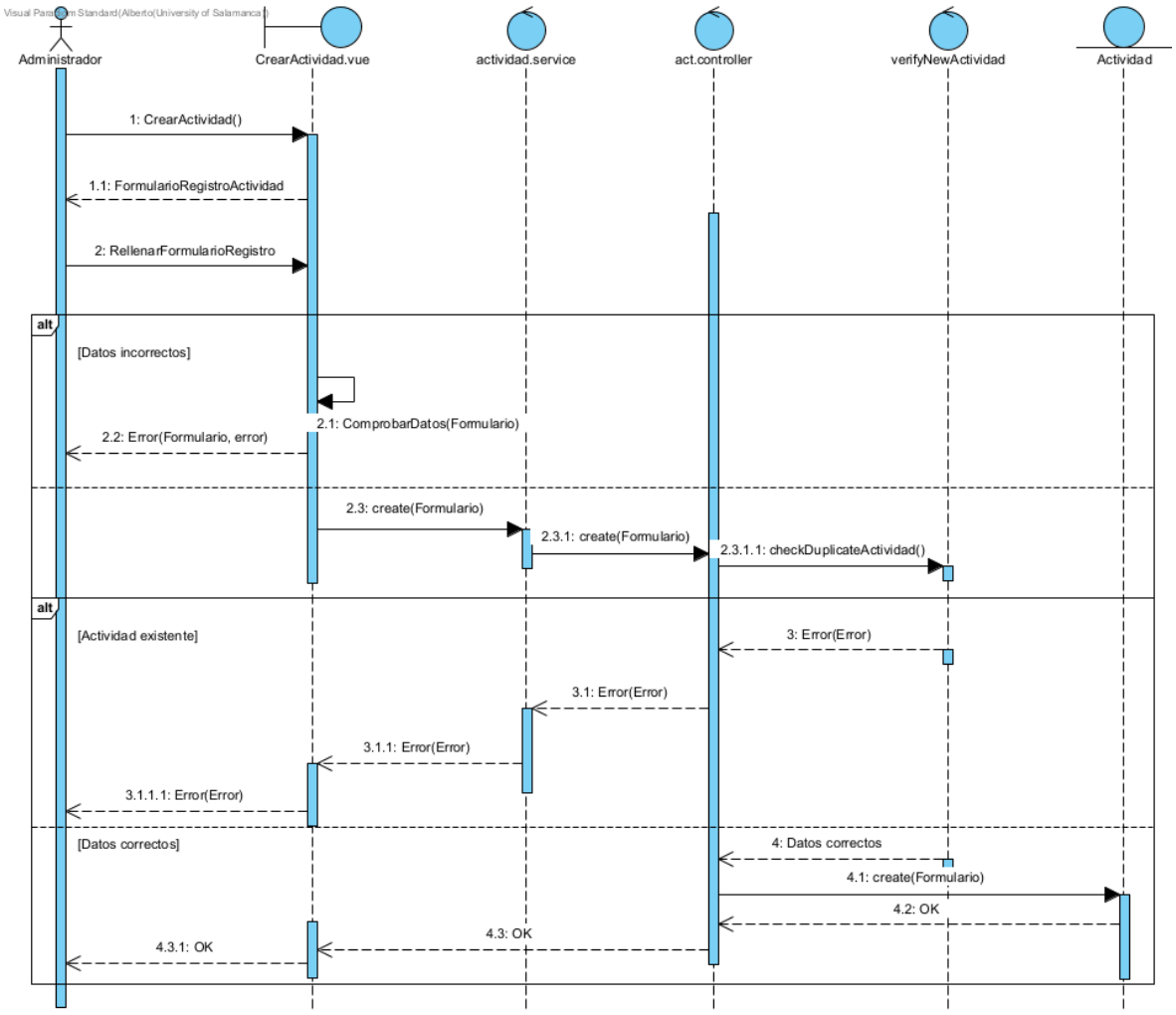

Figura 2.48: UC-0022 Crear actividad

2.5.5. Gestión de datos de usuario

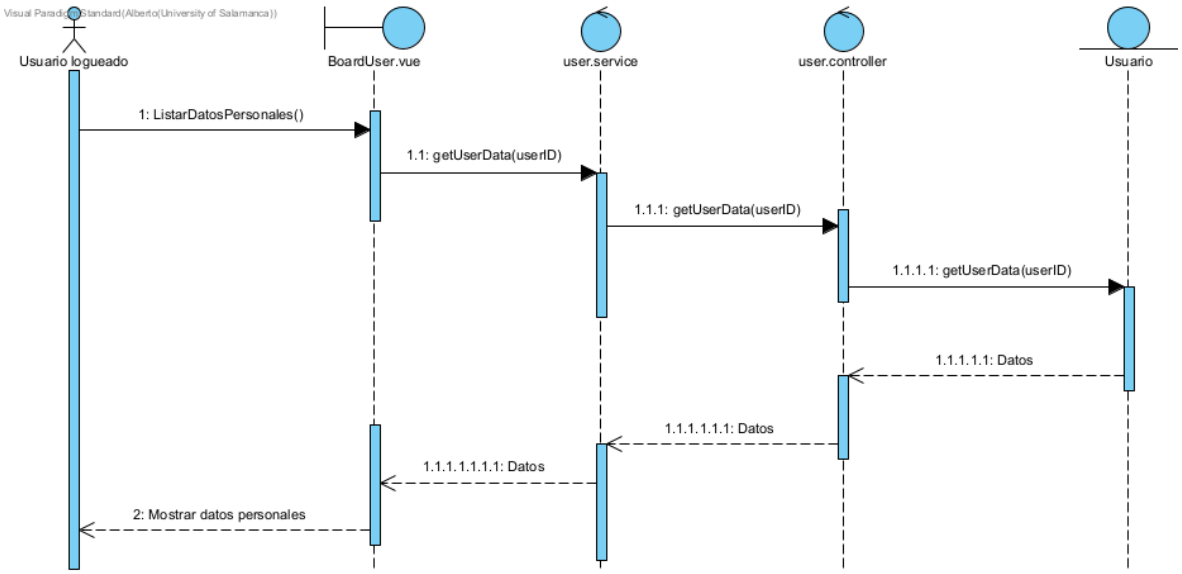

Figura 2.49: UC-0023 Listar datos personales

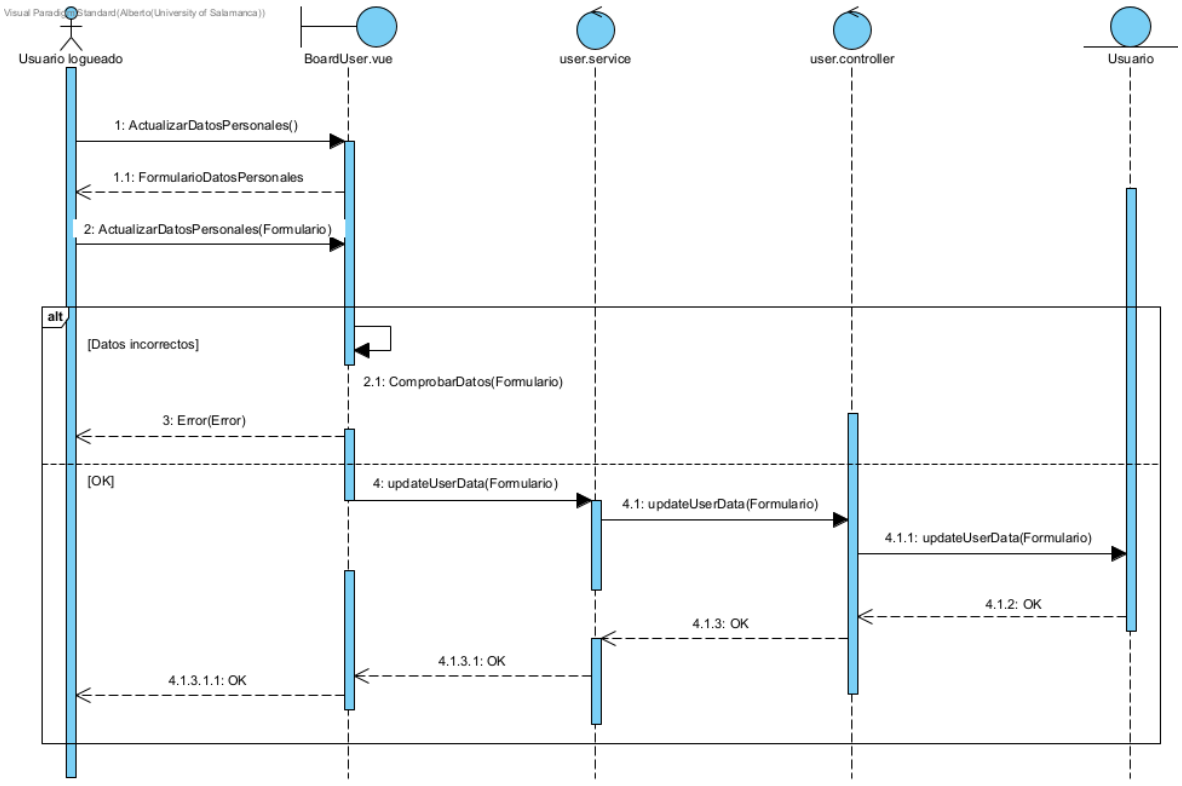

Figura 2.50: UC-0024 Actualizar datos personales

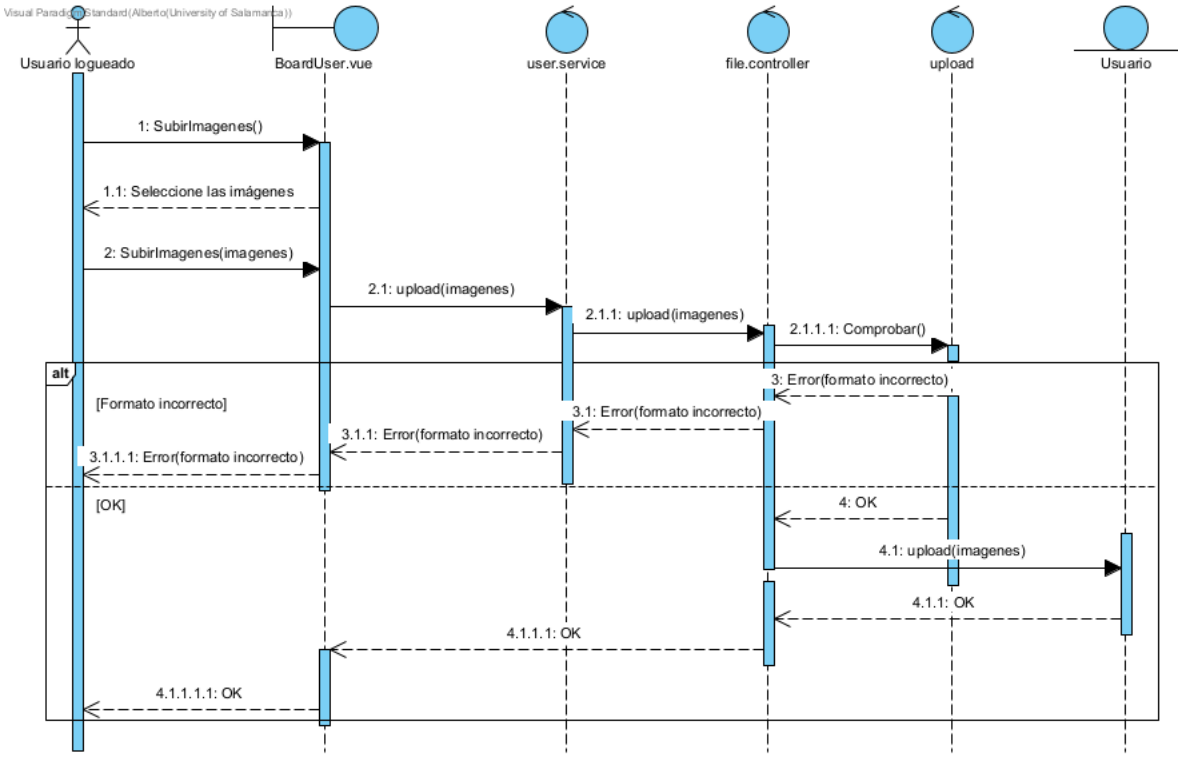

Figura 2.51: UC-0025 Subir imagen personal

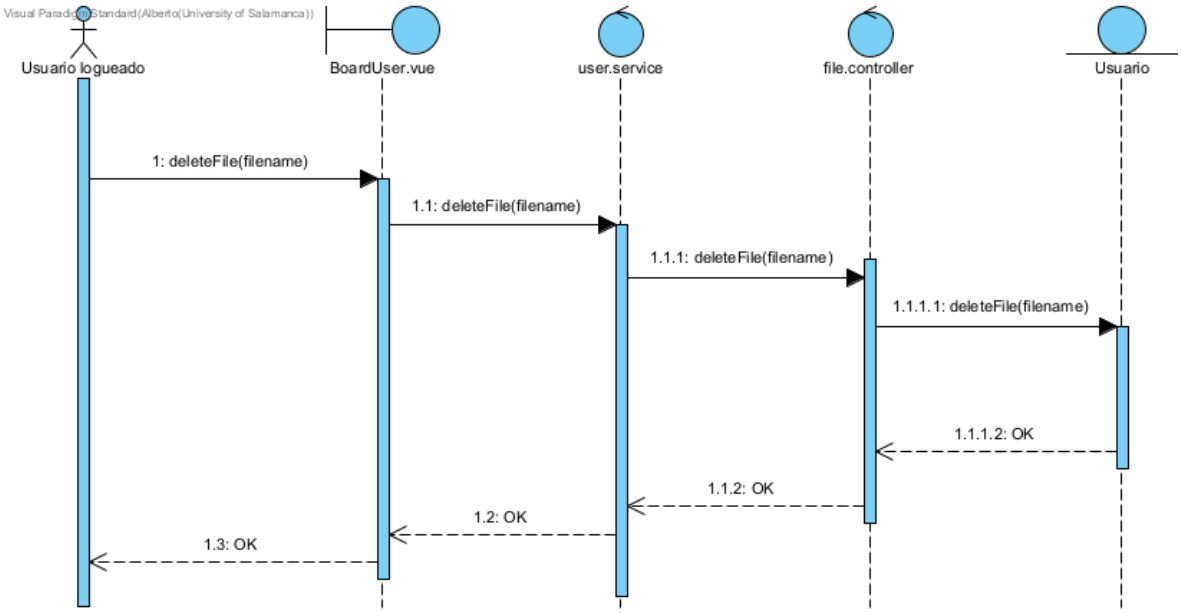

Figura 2.52: UC-0026 Borrar imagen personal

2.5.6. Gestión de gimnasio

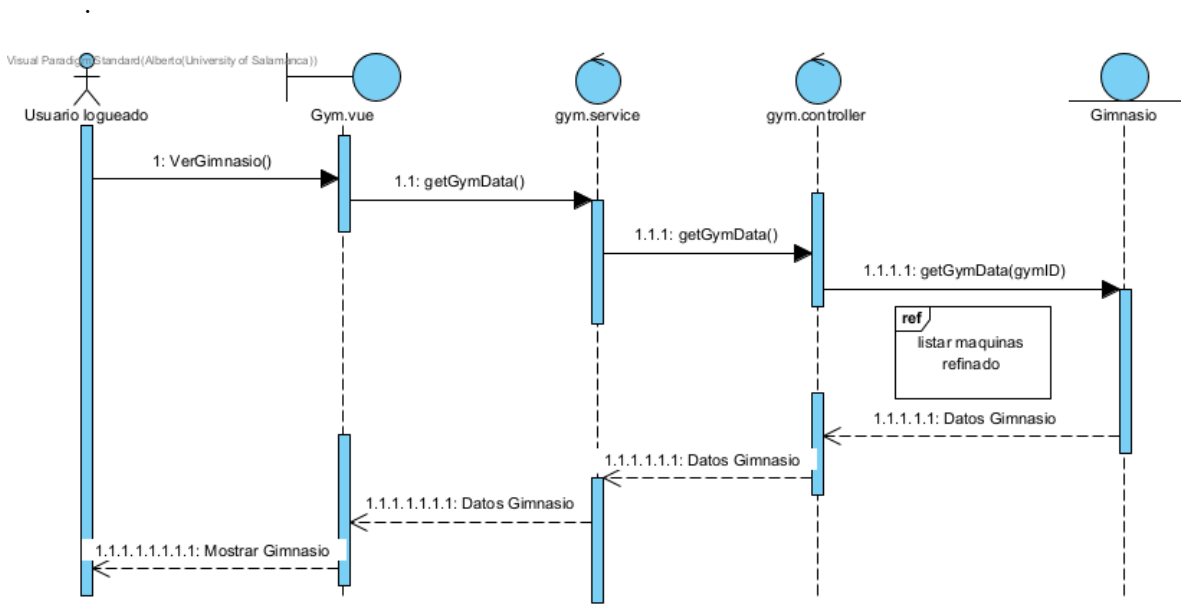

Figura 2.53: UC-0030 Ver gimnasio

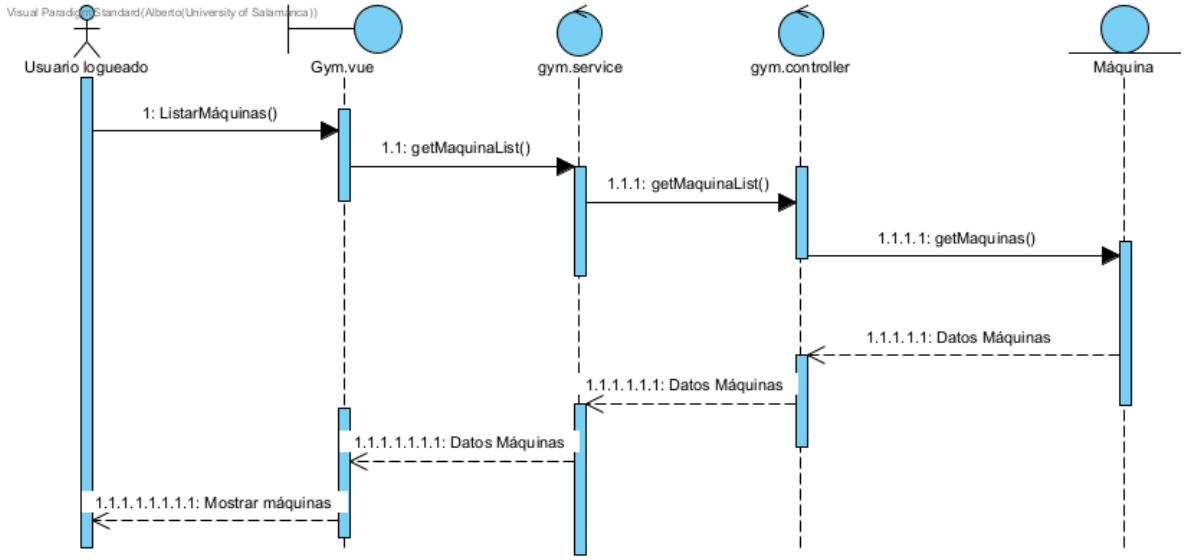

Figura 2.54: UC-0031 Listar máquinas

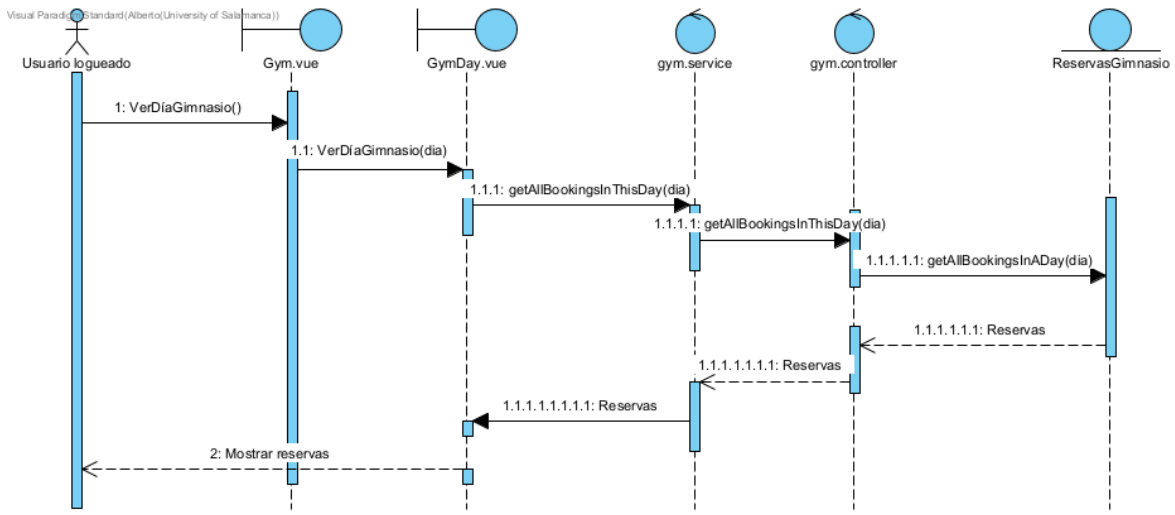

Figura 2.55: UC-0032 Ver día gimnasio

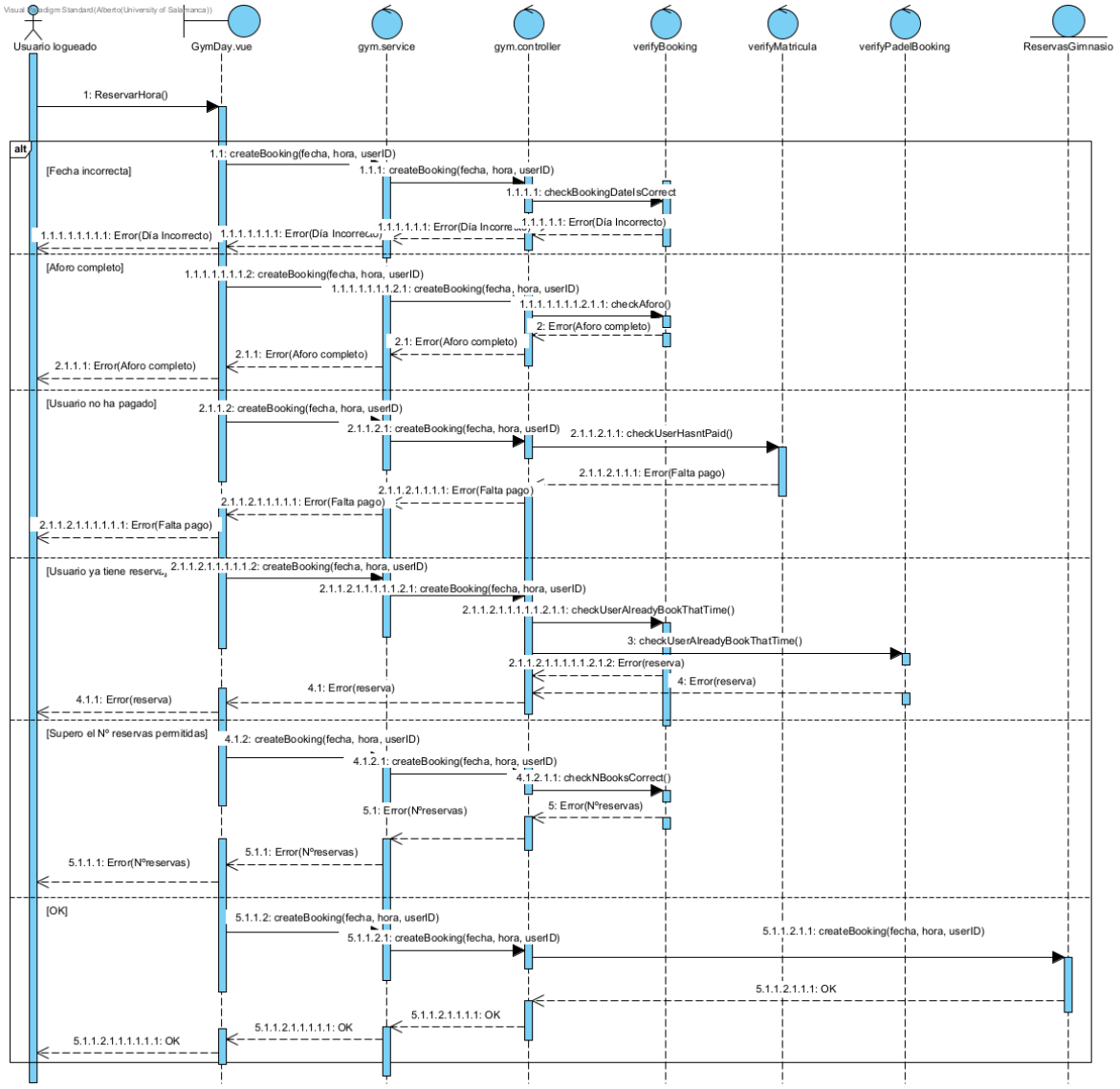

Figura 2.56: UC-0033 Reservar hora

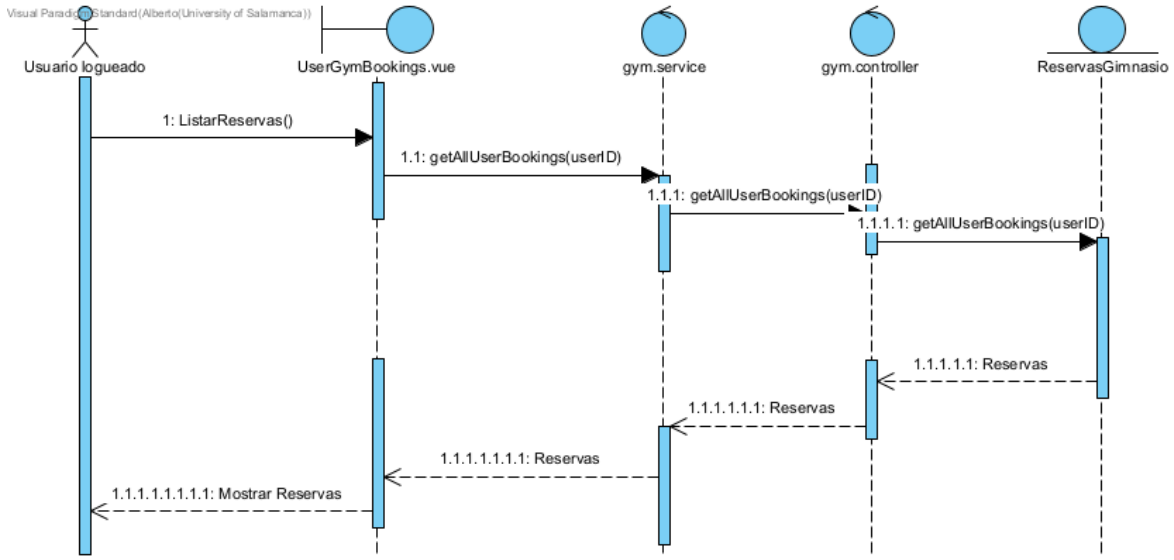

Figura 2.57: UC-0034 Listar reservas

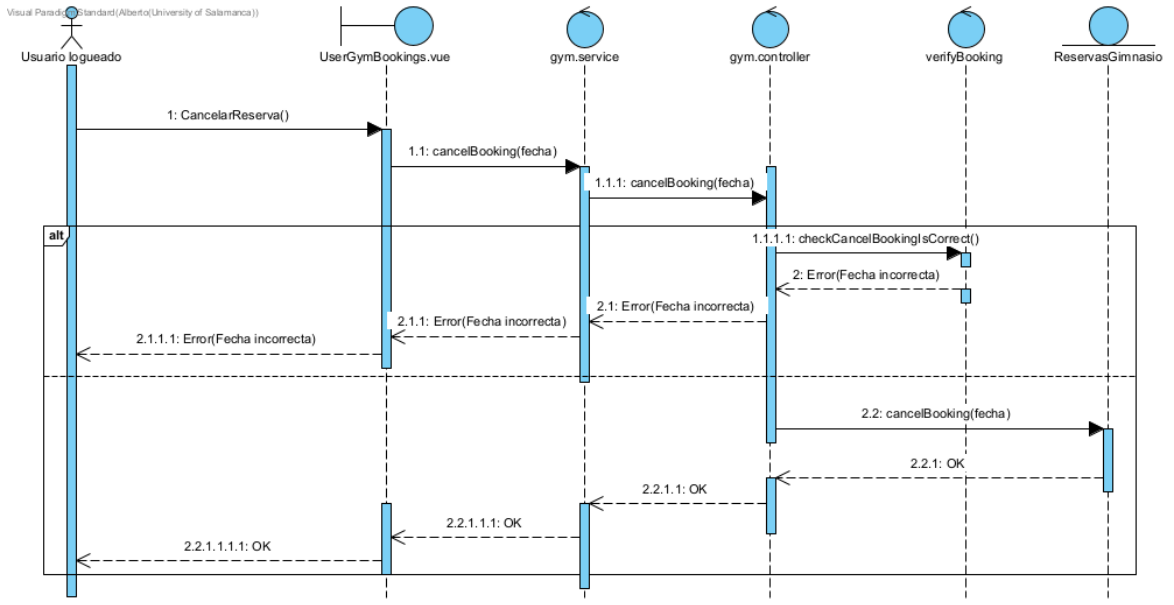

Figura 2.58: UC-0035 Cancelar reservas

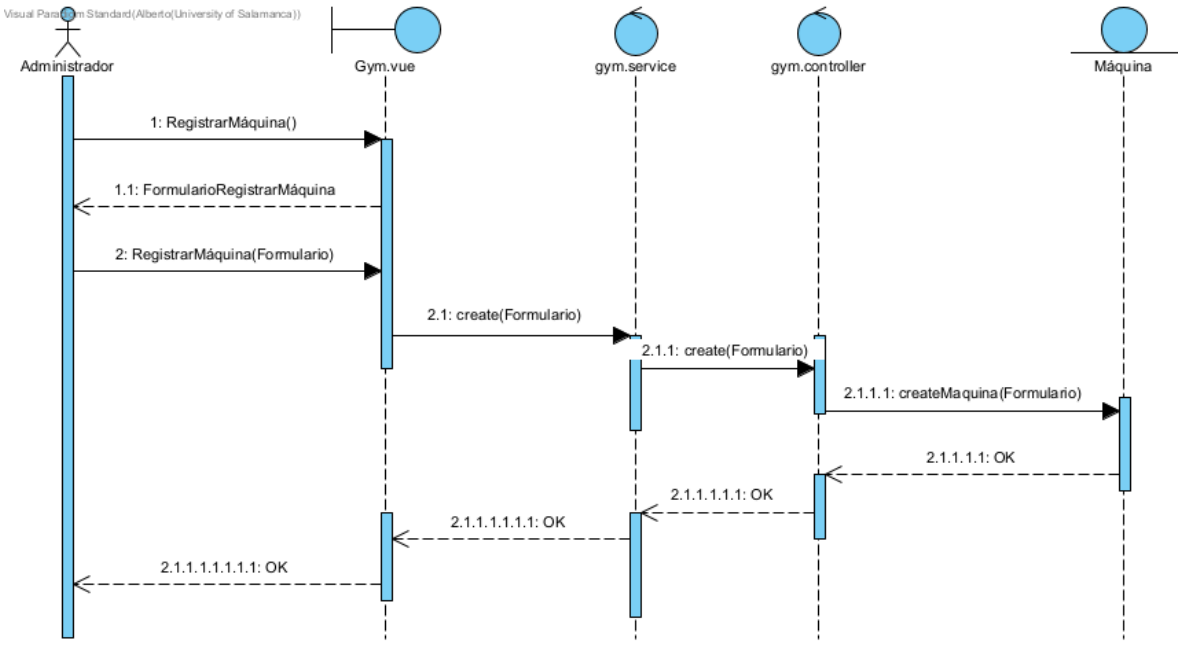

Figura 2.59: UC-0041 Registrar máquina

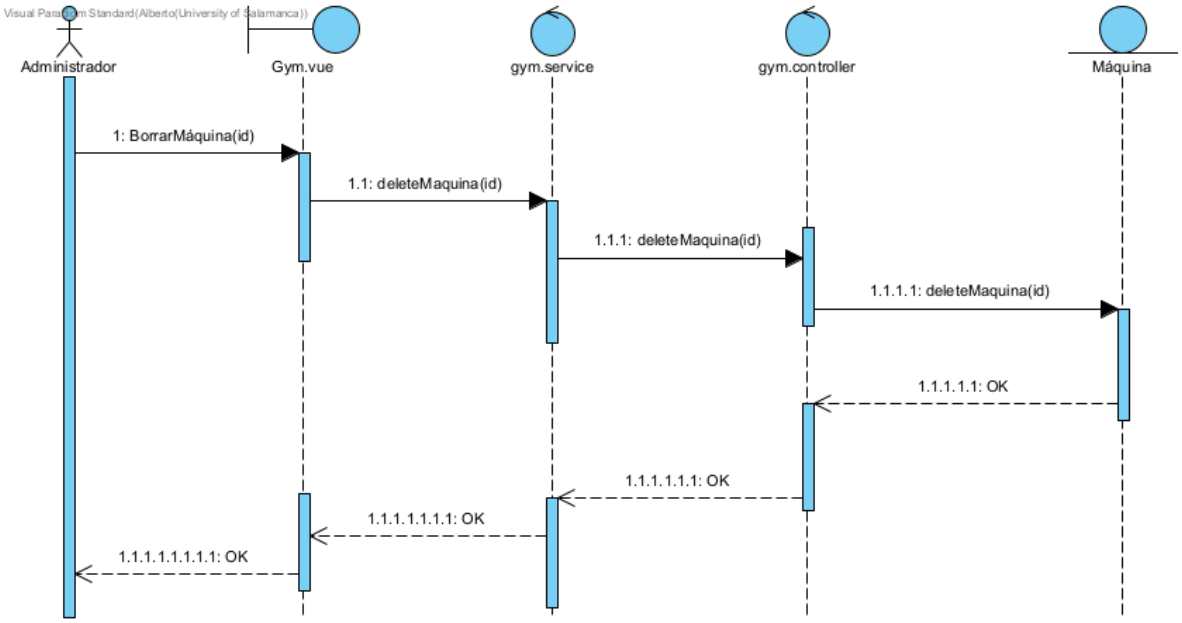

Figura 2.60: UC-0042 Borrar máquina

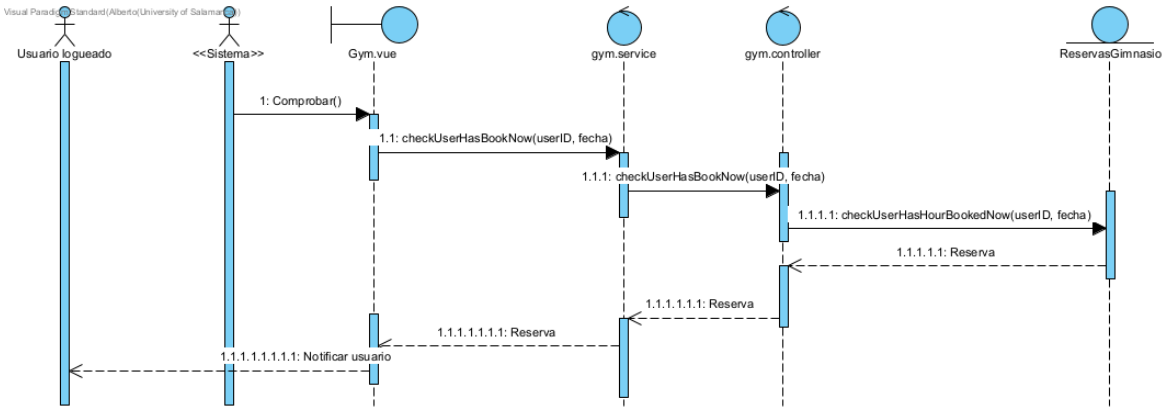

Figura 2.61: UC-0043 Comprobar usuario tiene hora

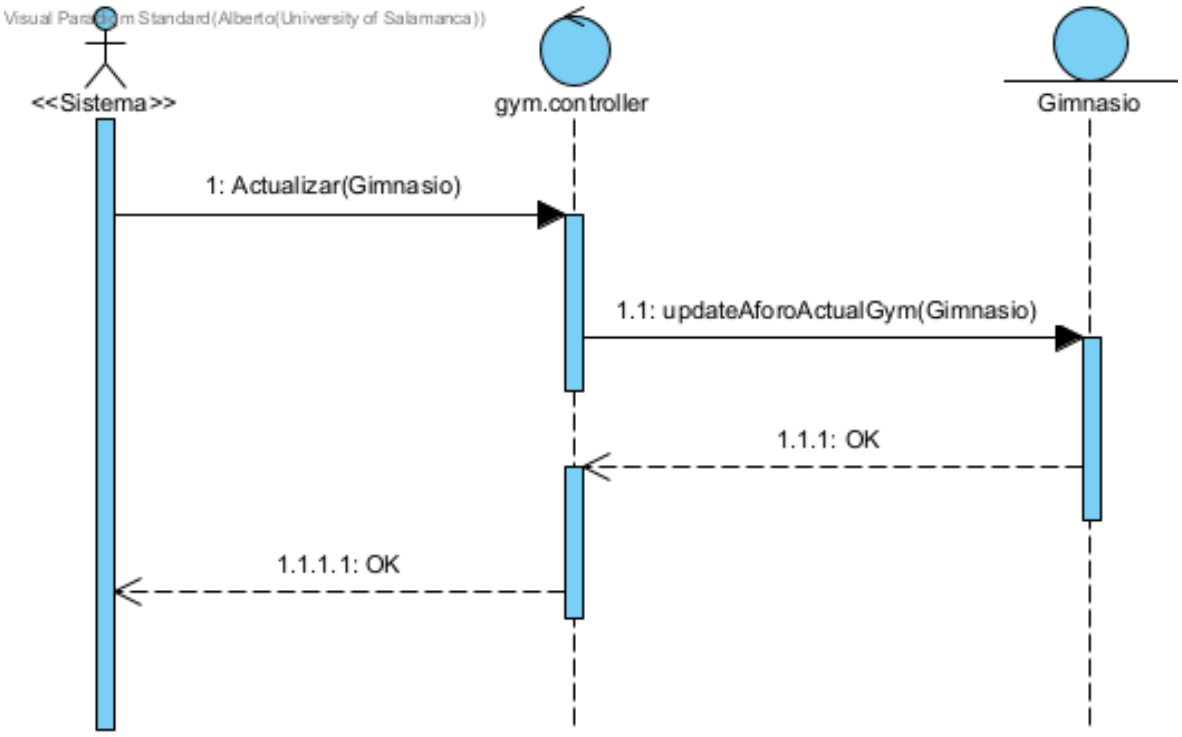

Figura 2.62: UC-0045 Actualizar aforo gimnasio

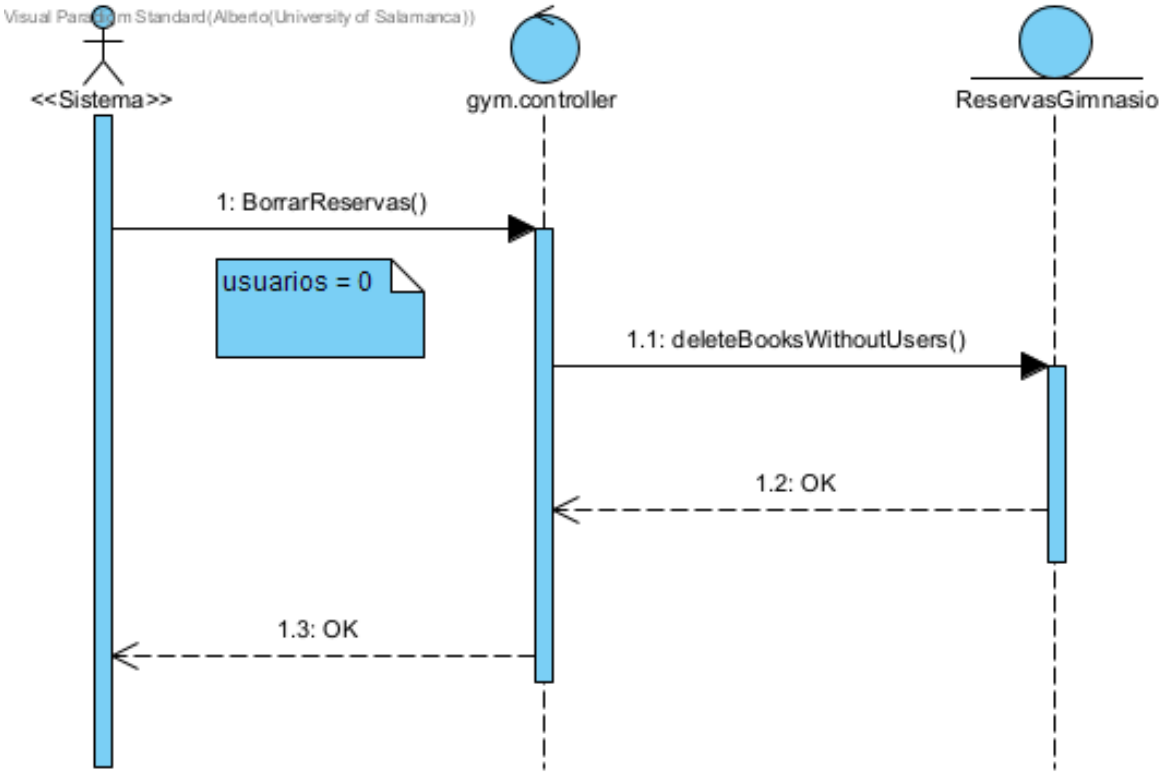

Figura 2.63: UC-0047 Borrar reservas canceladas gimnasio

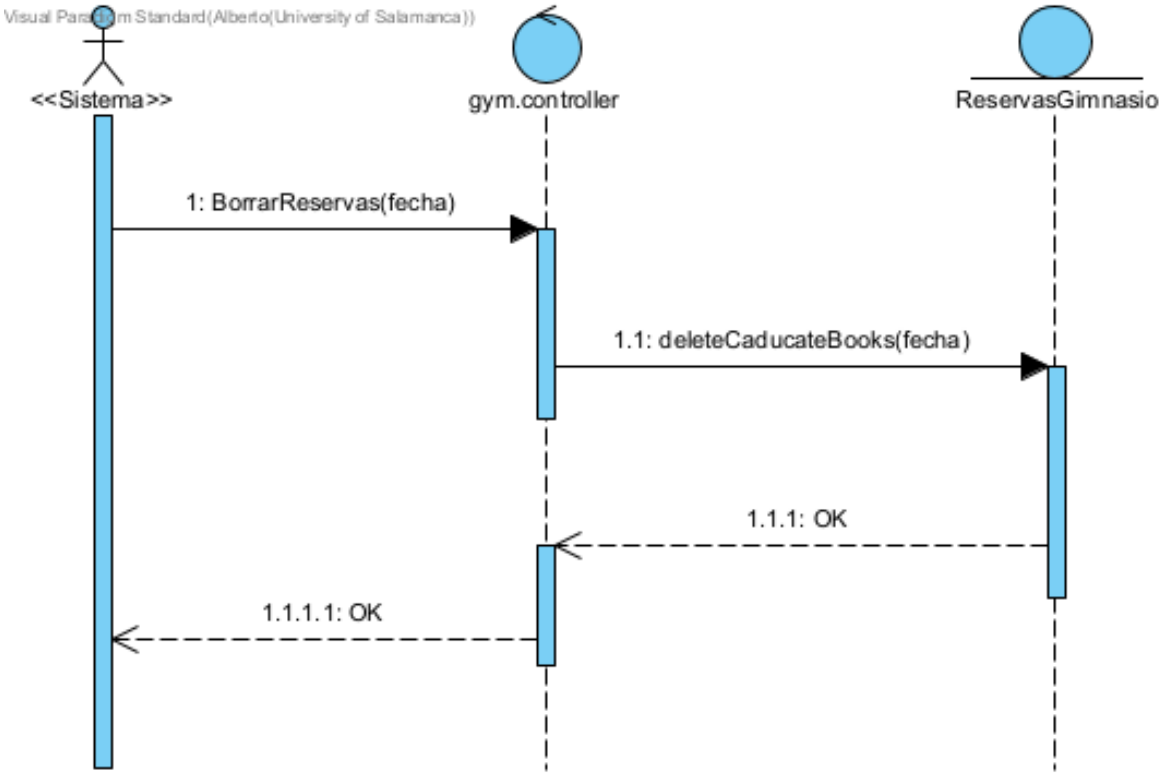

Figura 2.64: UC-0049 Borrar reservas caducadas gimnasio

2.5.7. Gestión de pádel

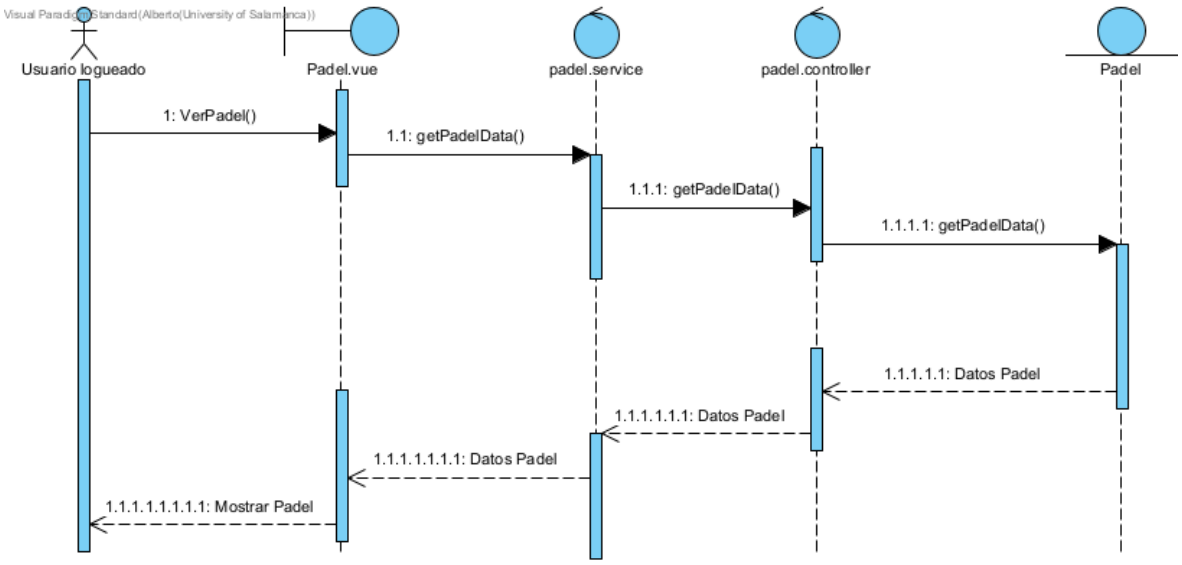

Figura 2.65: UC-0036 Ver pádel

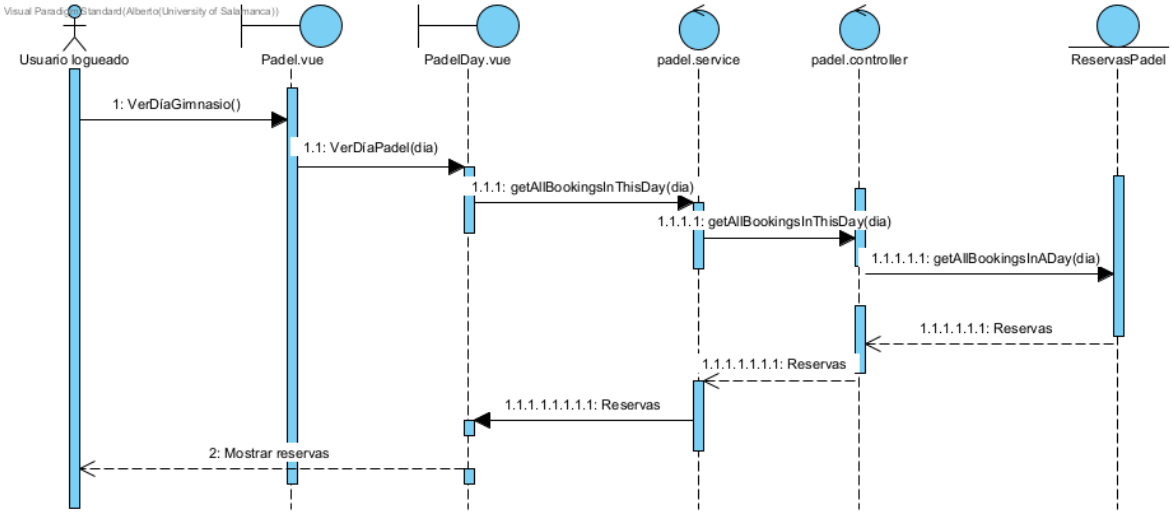

Figura 2.66: UC-0037 Ver día pádel

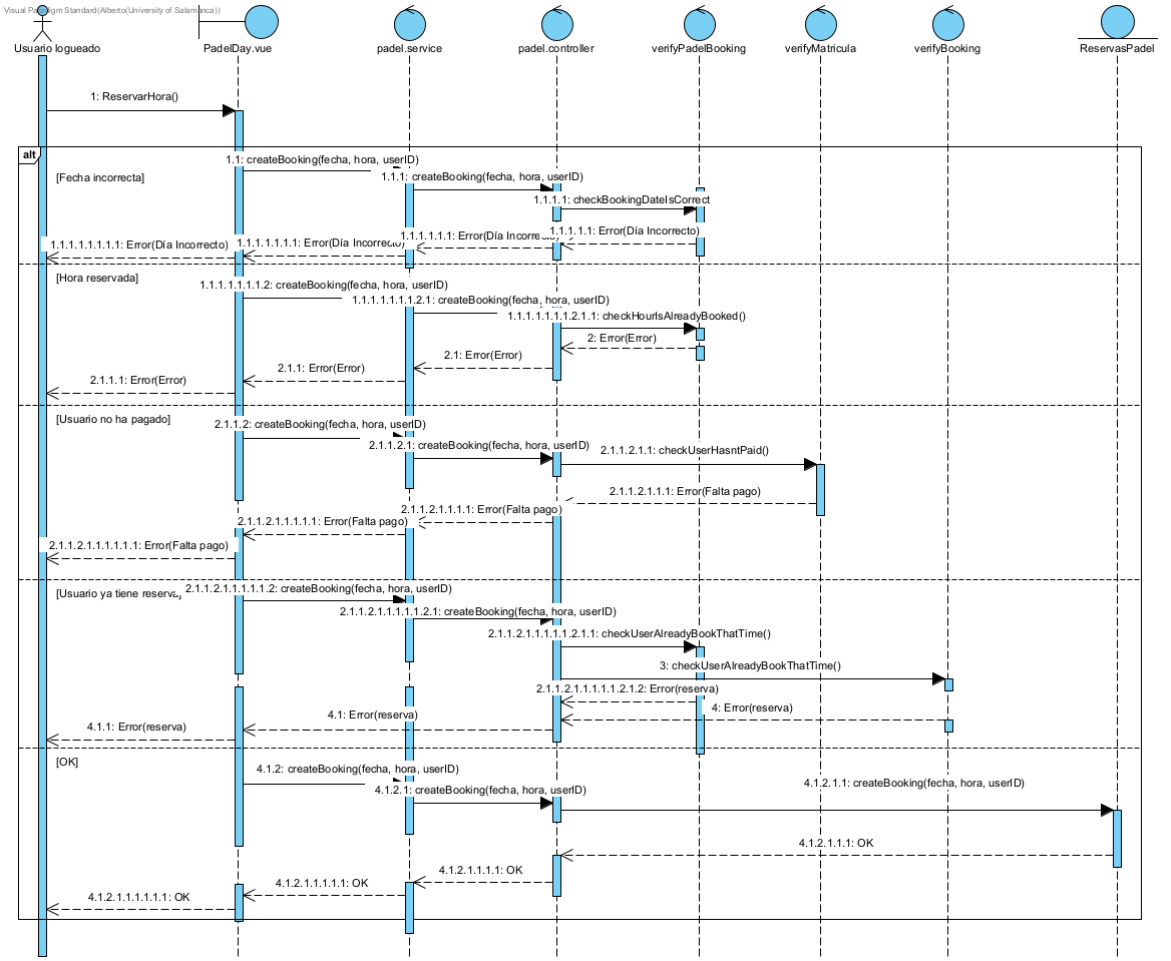

Figura 2.67: UC-0038 Reservar hora pádel

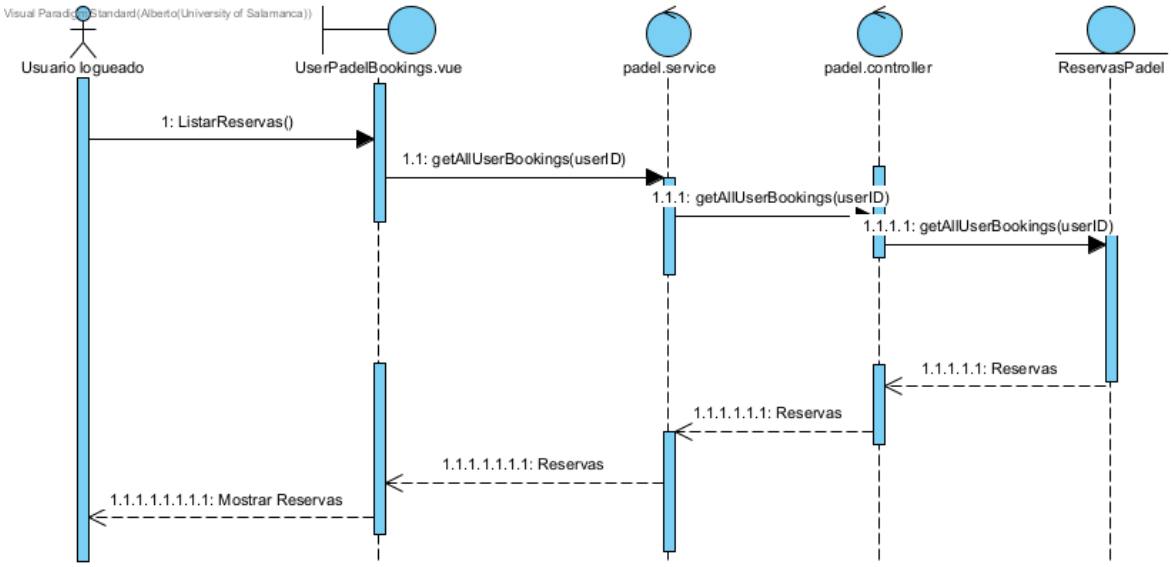

Figura 2.68: UC-0039 Listar reservas pádel

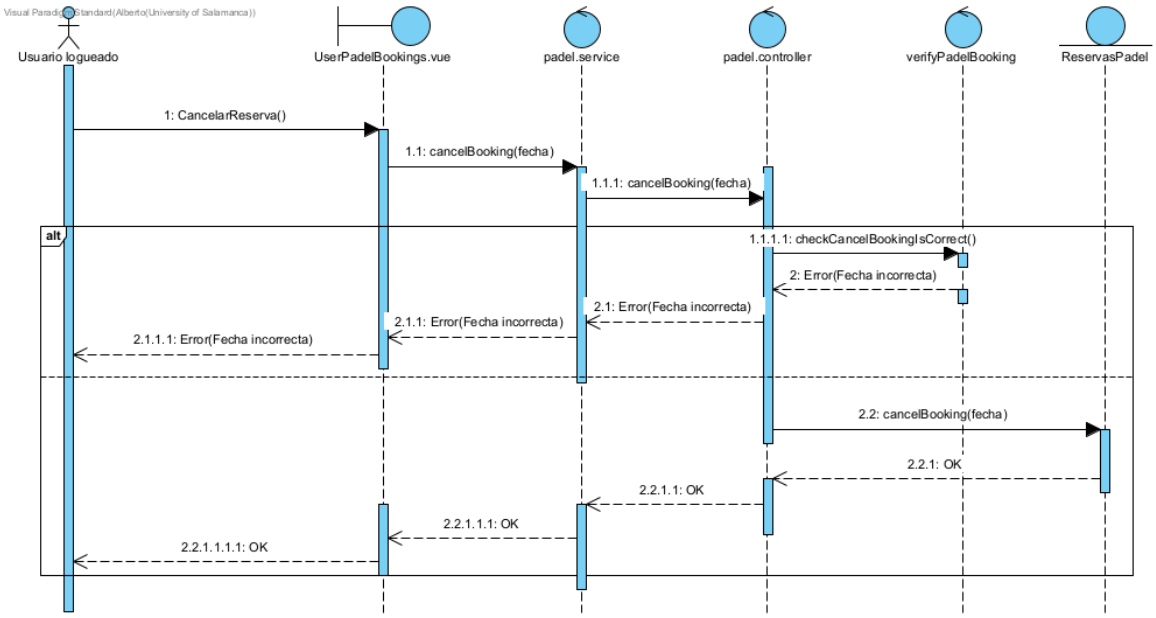

Figura 2.69: UC-0040 Cancelar reservas pádel

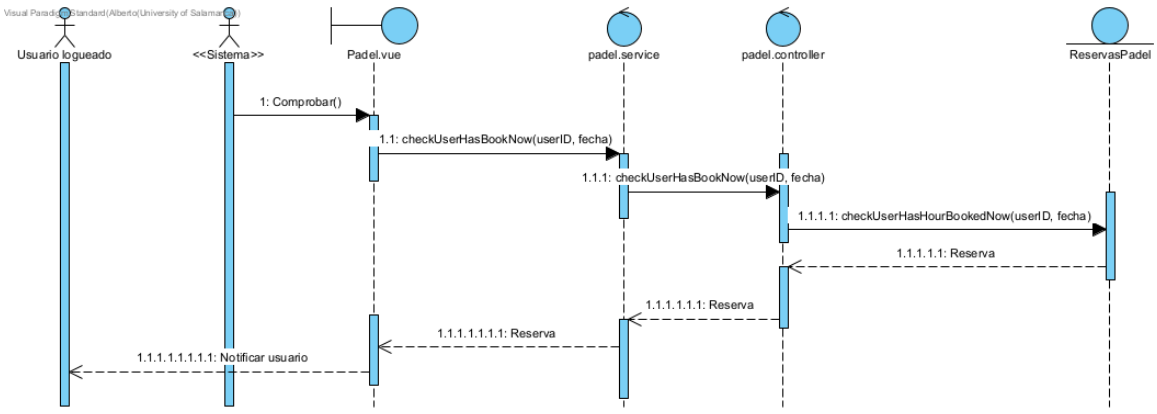

Figura 2.70: UC-0044 Comprobar usuario tiene hora pádel

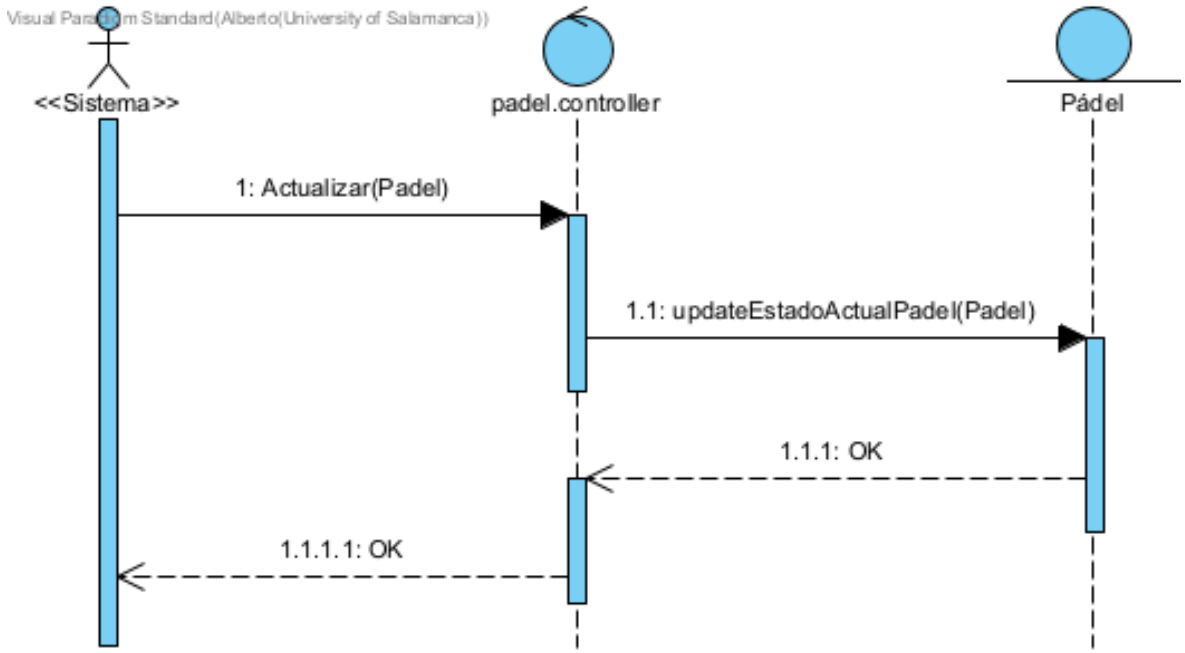

Figura 2.71: UC-0046 Actualizar estado pádel

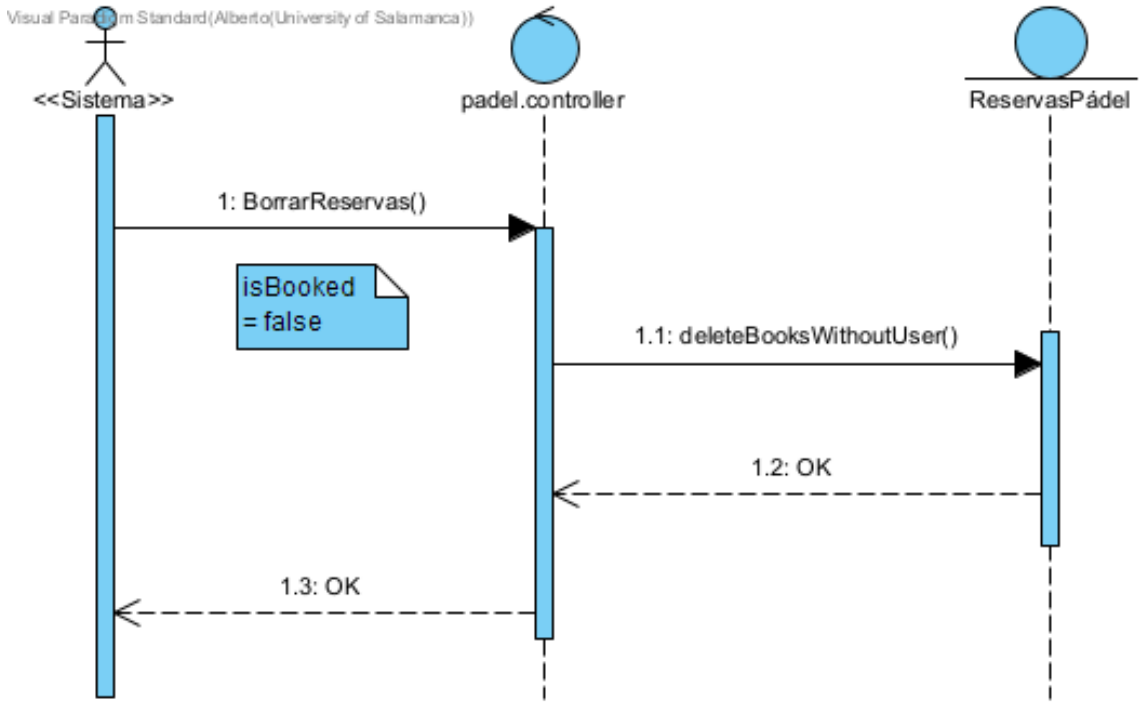

Figura 2.72: UC-0048 Borrar reservas canceladas pádel

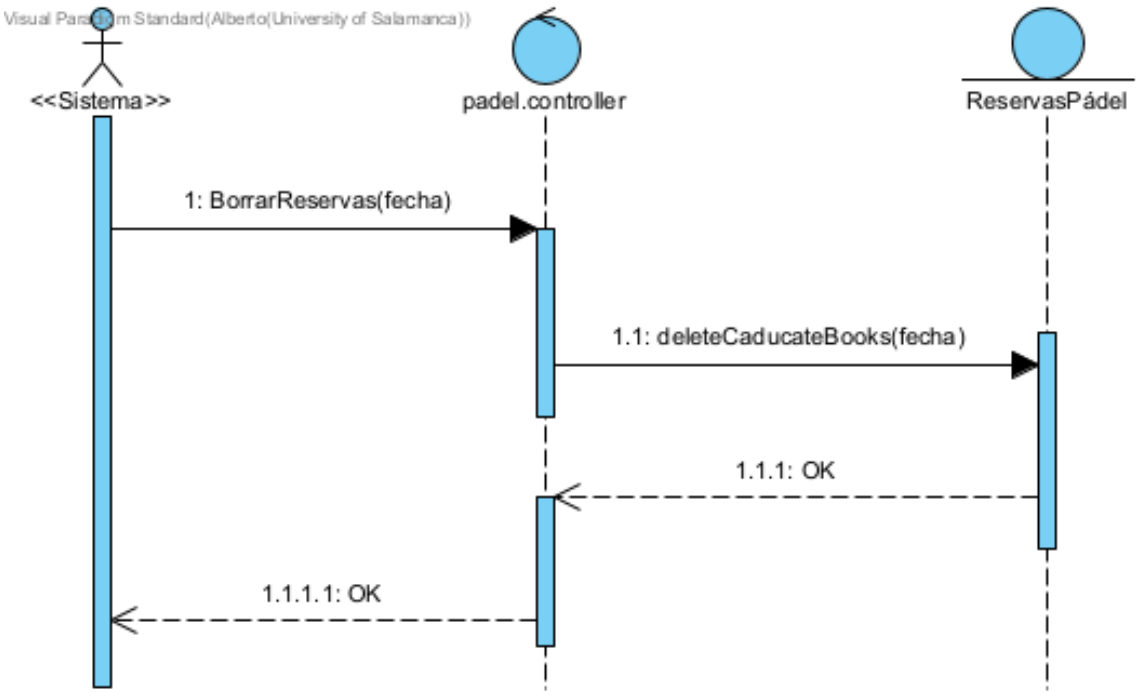

Figura 2.73: UC-0050 Borrar reservas caducadas pádel

3. Diseño de la base de datos

MongoDB es un sistema de base de datos NoSQL, orientado a documentos y de código abierto [1].

En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

MongoDB es una base de datos adecuada para su uso en producción y con múltiples funcionalidades.

Para el que el sistema pueda almacenar y gestionar la información necesaria, se usará una base de datos proporcionada por MongoDB, a continuación se muestra un ejemplo de un usuario registrado en la base de datos:


```
{
    "_id" : ObjectId("607723073a222f22785ecaf0"),
    "imagenes" : [
        "1618420645958-tfgym-actividad1.PNG",
        "1618420645949-tfgym-77140803-signo-del-corazón-el-pulso-y-el-icono-de-mancuerna-vector-.jpg",
        "1618420645954-tfgym-calen.PNG"
    ],
    "roles" : [
        ObjectId("605a2ad8fcbddc2f3cb9d803"),
        ObjectId("605a2ad8fcbddc2f3cb9d805")
    ],
    "actividades" : [
        ObjectId("606ed9445e5750330ca2af39")
    ],
    "nombre" : "Alberto",
    "apellidos" : "Martín",
    "email" : "alberto@gmail.com",
    "contraseña" : "$2b$10$5pHaXGw/Ef9nz4wNJiFLMOntyZTEK89l6q9tT.gxjHgemhsONTwbe",
    "matricula" : ObjectId("607723073a222f22785ecaef"),
    "datosPersonales" : {
        "peso" : 81,
        "altura" : 181,
        "imc" : 0.447513812154696
    },
    "nBooks" : 1,
    "__v" : 1
}
```


Figura 3.1: Ejemplo JSON de un usuario registrado

Como puntualización se explican todas las clases y las relaciones entre estas:

- **Usuario:** Es la colección que define a un usuario con todos sus datos que hace uso de la aplicación.
    - **Rol:** Esta colección define los roles que toman los usuarios, solo habiendo dos: Usuario y Administrador.
    - **ResetPasswordToken:** Esta colección define los tokens que se generan cuando un usuario solicita resetear su contraseña.

- **Matrícula:** Es la colección que define la matrícula que el usuario escoge para acceder a las funcionalidades del sistema.
- **Actividad:** Esta colección define las actividades creadas por los administradores y cursadas por los usuarios.
- **Reseña:** Esta colección contiene las opiniones de los usuarios sobre las actividades.
    - **VotosReseña:** En esta colección se almacena los likes y dislikes de diferentes usuarios a diferentes reseñas.
- **Padel:** Esta coleción contiene información sobre el pádel.
- **ReservasPadel:** En esta colección se definen todos los datos sobre las reservas para el pádel de los clientes.
- **Gimnasio:** Esta coleción contiene información sobre el gimnasio.
    - **Máquinas:** Es la colección que define las máquinas registradas en el sistema.
- **ReservasGimnasio:** En esta colección se definen todos los datos sobre las reservas para el gimnasio de los clientes.

4. Modelo de despliegue

En el siguiente diagrama se puede observar el despliegue del sistema:

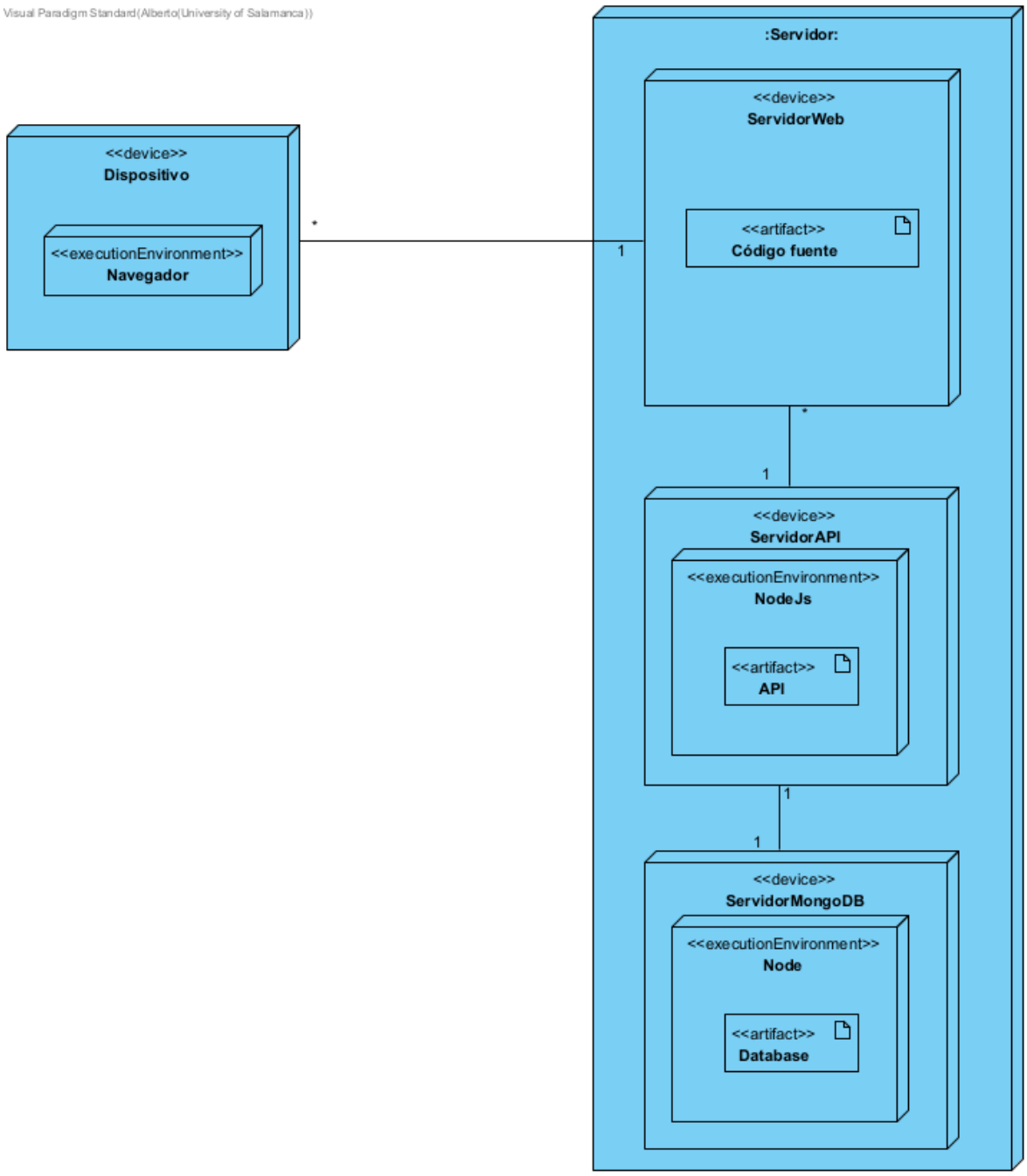

Figura 4.1: Diagrama de despliegue

Hay 2 nodos principales:

- **Dispositivo:** Dispositivo donde se encuentra un navegador. Mediante este na-

vegador se podrá acceder a la aplicación donde los usuarios podrán realizar las distintas acciones del sistema detalladas anteriormente.

- **Servidor:** Que se dividie a su vez en tres nodos:
  - **Servidor Web:** Servidor en el que se almacenan los datos necesarios para desplegar la aplicación web en el cliente web. Se comunicará con el servidor API para acceder a la base de datos y realizar la autenticación de usuarios
  - **Servidor API:** Servidor que, mediante la API, prestará soporte a los nodos dispositivo para el manejo de la información del sistema. Cabe destacar que el entorno de ejecución se trata de Node.js.
  - **Servidor MongoDB:** Servidor que actúa como base de datos y donde se almacenará la información necesaria del sistema. La base de datos será MongoDB.

5. Bibliografía

[1] Mongodb. URL https://sitiobigdata.com/2017/12/27/mongodb-arquitectura-y-modelo-de-datos/.

[2] Visual paradigm. URL https://www.visual-paradigm.com/support/documents/vpuserguide.jsp.

[3] Vue.js, . URL https://es.vuejs.org/v2/guide/.

[4] Vuex, . URL https://vuex.vuejs.org/.

[5] S. McConnell. "desarrollo y gestión de proyectos informáticos". In *Mc Graw Hill*, 1997.

[6] R. S. Pressman. "ingeniería del software: Un enfoque práctico". In 7^a *Edición*, 2010.