Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de una aplicación web para el control de acceso a un autobus escolar basado en IoT

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Ignacio José Serra Almenar

Tutor: Lenin G. Lemus Zúñiga

2016/2017

Resumen

El objetivo de este proyecto es la creación de una aplicación web que permita gestionar la entrada y la salida de los alumnos de un colegio a un autobús.

La aplicación incluye como gestión la capacidad de creación, modificación, eliminación y listado de autobuses, conductores, monitores, rutas, alumnos, cursos, parientes y paradas, así como la opción de enviar mails informativos y automáticos a los parientes si en una determinada ruta ha ocurrido un atasco. También permite enviar notificaciones de Telegram informando a los parientes del estado del alumno.

Además, cuenta con una vista pública en la que los parientes de un alumno pueden consultar el estado de sus hijos, así como la posición en *Google Maps*. Con este servicio se espera informar a los parientes de los alumnos de la llegada de sus hijos antes de que ésta tenga lugar.

Se destaca que con la librería Semantic UI, la interfaz de la aplicación web se adapta a móviles y tabletas.

Palabras clave: Aplicación Web, Internet of Things, GPS, JavaScript/jQuery, Node.js, MySQL, AJAX, Jade, Semantic UI

Abstract

The objective of this project is the creation of a web application that allows to manage the entry and exit of students from a school to a bus.

The application includes the ability to create, modify, delete and list buses, drivers, monitors, routes, students, courses, relatives and stops, as well as the option to send informative and automatic emails to relatives if a Route has a traffic jam. It also allows you to send Telegram notifications reporting relatives of the student's status.

Furthermore, it has a public view in which the relatives of a student can check the status of their children, as well as the position in Google Maps. This service is expected to notify the relatives of students of the arrival of their children before it takes place.

It should be noted that with the library Semantic UI, the web application interface fits mobile and tablets.

Keywords: Application Web, Internet of Things, GPS, JavaScript/JQuery, Node.js, MySQL, AJAX, Jade, Semantic-UI.

Tabla de contenidos

Índice de figuras

1. Introducción

La plataforma descrita en este trabajo pretende solventar la problemática existente a la hora de efectuar la recogida de un alumno que use transporte escolar.

En primer lugar, se dispone de un colegio con alumnos divididos en diferentes grupos. Los alumnos están inscritos por curso escolar. De forma que se va a tener alumnos de infantil1, infantil2, primero, segundo, tercero, cuarto, quinto y sexto.

Cada grupo va a ir destinado a un autobús, el cual tiene su propia línea y su propio horario.

Inicialmente se van a utilizar cinco líneas en este problema con opción a poder ser ampliadas.

Cada línea va a tener asignada un conjunto de paradas, en las cuales se indica posición, tiempo estimado de llegada y tiempo estimado de partida.

El autobús dispone de un monitor, el cual va a llevar una lista electrónica de alumnos pertenecientes a su grupo.

Cuando los alumnos se dirijan al autobús, el monitor va a estar presente para controlar la cantidad de alumnos que hay.

Cada vez que éstos vayan subiendo al autobús, ficharán la entrada en la lista electrónica del monitor para aparecer como registrados en la aplicación.

De esta manera, en sus datos registrados aparecerá la entrada del fichaje, por lo que utilizando la aplicación web los parientes del alumno sabrán que está dentro del autobús. También podrán comprobar la ruta que seguirá y las paradas establecidas.

En cada parada, los alumnos que bajen ficharán la salida. Al realizar este último fichaje, el monitor se fijará en el listado de parientes del alumno, donde aparecerán las fotos de las personas encargadas de recoger al alumno. El monitor se encargará de controlar que el pariente que recoge al alumno en la parada se corresponde con alguno de los tutores que aparecen en la pantalla.

Si se diera el caso de que no ha llegado el pariente que debe de recoger al alumno, el autobús seguirá su ruta con el alumno dentro. Si el pariente no llega a tiempo, mediante la aplicación web en su móvil puede comprobar el estado del alumno y adelantarse al autobús en cualquiera de las siguientes paradas.

Una vez la persona encargada recoge al estudiante, se almacena que el estudiante se dejó a cargo de la persona tutora, así como la fecha.

La última parada de todos los autobuses será el colegio.

1.1 Objetivos

Los objetivos del proyecto son los siguientes:

- Ofrecer un servicio que permita gestionar la creación, modificación y eliminación de autobuses, conductores, monitores, alumnos, parientes de alumnos, paradas, rutas, y cursos escolares.
- Permitir a los parientes conocer el estado en el que se encuentra su hijo mediante la aplicación web o a través de notificaciones *push* en grupos de Telegram.
- Capacidad de avisar por correo electrónico a los padres de los alumnos si una ruta ha sufrido un atasco y va a retrasar la llegada del alumno.

Con estos objetivos se pretende que el colegio encargado gestione su servicio de autobuses escolares ofreciendo a los parientes información constante y automatizada del estado de los alumnos.

1.2 Contenido de la memoria

La memoria del proyecto va a contener las diferentes fases del desarrollo que se han seguido para la creación de la aplicación web:

- **Búsqueda bibliográfica**: localización de aplicaciones similares para estudiar y analizar su estructura.
- **Análisis de requerimientos**: requisitos necesarios de la aplicación web, distinguiendo entre requisitos funcionales y no funcionales.
- **Diseño de la aplicación**: casos de uso de la aplicación, diagrama de clases, diagrama de la base de datos, prototipo de las vistas y versión final.
- **Implementación**: lenguaje de programación y código escrito para la realización de la aplicación web.
- **Validación y pruebas**: pruebas realizadas y comprobaciones de la aplicación web.
- **Conclusiones**: funcionalidades que se han hecho, que no se han podido y que se podrían hacer en un futuro.
- **Bibliografía**: libros y páginas utilizadas para el estudio y realización del proyecto.

2. Búsqueda bibliográfica

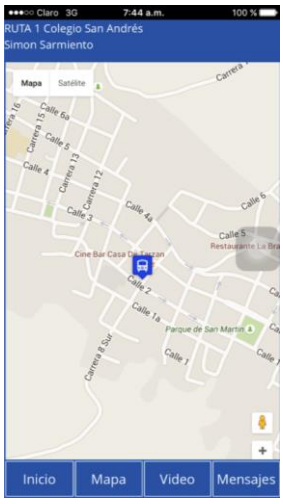
El primer paso para el desarrollo del proyecto es la búsqueda de aplicaciones ya existentes y similares con el fin de poder conocer su estructura y lo que ofrecen.

Tu Ruta Escolar

En la Ilustración 1 se observa que la aplicación utiliza un diseño minimalista y *responsive*[1]. Una vez registrados y con la sesión iniciada, la primera vista disponible nos indica que tenemos que vincular la ruta donde se encuentra el alumno el cual queremos comprobar el estado.

Una mejora podría ser la vinculación por alumno, ya que de este modo el pariente no necesitaría aprenderse a qué ruta está destinado su hijo.

Ilustración 1: App Tu Ruta Escolar - Selección de Ruta



Una vez vinculada la ruta, nos muestra la posición del autobús por GPS en vista de mapa o de satélite, la posibilidad de ver vídeo por *streaming* del autobús y de mandar mensajes de texto a los encargados del éste.

En definitiva, es una aplicación de monitorización constante para los parientes de los alumnos.

Ilustración 2: App Tu Ruta Escolar – GPS

[1] *Responsive:* Técnica de diseño web que busca la correcta visualización de una misma página en distintos dispositivos.

Traceus

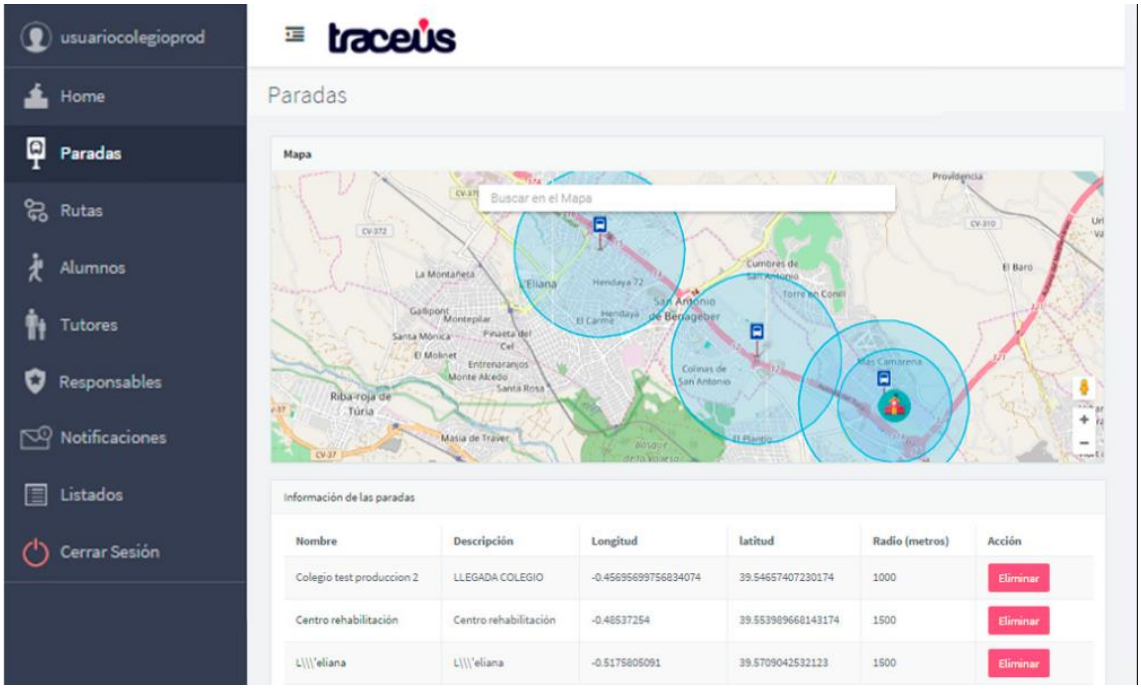


Ilustración 3: Traceus App – Gestión

Ilustración 4: Traceus App - GPS

Traceus es una aplicación que permite gestionar un servicio escolar mediante funciones CRUD[2] y además proporcionar a los parientes notificaciones de la posición de sus hijos. Tiene un diseño *responsive* y específico para móvil (Ilustración 2 y 4) y web (Ilustración 3).

[2] CRUD: (Create, Read, Update and Delete) Operaciones básicas que se realizan en una base de datos.

De las dos aplicaciones, la que más se va a asemejar al proyecto es la de Traceus, ya que va a tener un servicio de administración similar, con la inclusión de un sistema de guardado de imágenes, que permitirá reconocer al pariente que recoja a su hijo, y un envío de notificaciones de estado a los padres mediante la aplicación de Telegram.

3. Análisis de requerimientos

3.1 Introducción

3.1.1 Propósito

La especificación de requisitos tiene como finalidad conocer que se espera obtener de la aplicación, definiendo de manera clara y exacta las funcionalidades y restricciones que presentará la aplicación que se quiere desarrollar.

3.1.2 Ámbito del Sistema

La aplicación web a desarrollar va a ser una plataforma que permite la gestión de autobuses escolares a encargados de un colegio, y la consulta por parte de parientes del estado en el que se encuentran sus hijos.

3.1.3 Definiciones, Acrónimos y Abreviaturas

Alumno: Todas las personas matriculadas en el colegio.

Monitor: Persona encargada de la entrada y salida de alumnos matriculados al autobús asignado.

Conductor: Persona que conduce el autobús designado y se encarga de efectuar las paradas pertinentes para recoger o dejar a los alumnos.

Parada: Lugar donde los alumnos matriculados esperan la llegada de su autobús designado o puntos de destino donde se reúnen parientes para recoger a sus hijos.

Centro: Lugar donde se imparten los diferentes niveles de educación.

Rutas: Conjunto de paradas que sigue el autobús con el fin de llevar a los alumnos a su destino.

Navegador: Visualizador de páginas web a través de internet que permiten al usuario realizar consultas o gestiones.

Servidor web: Programa informático que procesa una aplicación desde el lado del servidor [5], realizando conexiones con el cliente y generando respuestas.

Servidor Apache: Servidor HTTP de código abierto para plataformas Unix, Linux, Windows y Macintosh [7].

MySQL: Sistema de gestión de bases de datos relacional.

JavaScript: Lenguaje de programación interpretado, débilmente tipado y dinámico [8].

Node.js: Entorno en tiempo de ejecución multiplataforma para JavaScript, de código abierto [9].

CSS: Hojas de estilo en cascada, lenguaje formal de ordenador usado para definir la presentación de un documento estructurado en HTML o XML [11].

Framework: Estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado. Facilitan el desarrollo y evitan los detalles de bajo nivel [6].

Semantic UI: *Framework* utilizado en aplicaciones y páginas web que ayuda a crear entornos *responsive* y estilizados.

Jade: Lenguaje de plantillas centrado en la escritura de código rápida de HTML [13].

HTML: Lenguaje de marcas de hipertexto para la elaboración de páginas web [14].

AJAX: *Asynchronous JavaScript And XML*, es una técnica de desarrollo web para crear aplicaciones interactivas [15].

Internet of Things: Es un concepto que se refiere a la interconexión digital de objetos cotidianos con internet. [2]

Telegram: Servicio de mensajería instantánea.

3.1.4 Referencias

- IEEE Std. 830-1998 Guía del IEEE para la Especificación de Requisitos Software.
- Ejemplos IEEE

3.1.5 Visión General del Documento

El documento se desglosa en dos secciones. “La Descripción General”, que va a tratar de aquellos factores que afectan al producto a construir, dando a conocer sus funciones principales, datos requeridos y restricciones que afecten a su desarrollo. “Los Requisitos Específicos”, cuyo contenido va a ofrecer un nivel de detalle suficiente como para permitir crear un sistema que satisfaga los requisitos.

3.2 Descripción General

3.2.1 Perspectiva del Producto

La aplicación web es totalmente independiente.

Las funciones varían dependiendo del tipo de usuario.

Los parientes van a poder realizar consultas, y los administradores gestionar y consultar

La plataforma podrá ser visitada por cualquier usuario independientemente del navegador que utilice, sistema operativo o dispositivo mientras esté disponible la conexión a Internet.

3.2.2 Funciones del Producto

Las funciones que va a satisfacer la aplicación web son las siguientes.

Usuario

- Ingreso de DNI.
- Consulta de estado de alumnos a su cargo.
- Consulta de mapa con coordenadas de las paradas.

Administrador

- Creación, modificación y eliminación de autobuses.
- Creación, modificación y eliminación de conductores.
- Creación, modificación y eliminación de monitores.
- Creación, modificación y eliminación de rutas.
- Creación, modificación y eliminación de rutas de ida.
- Creación, modificación y eliminación de rutas de vuelta.
- Creación, modificación y eliminación de paradas.
- Creación, modificación y eliminación de cursos escolares.
- Creación, modificación y eliminación de matriculados.
- Filtros en cada tabla por campos relevantes.
- Envío de correos electrónicos automáticos a los parientes que tienen un hijo cuya ruta ha sufrido un atasco.
- Envío de notificaciones por Telegram del estado de los alumnos a los parientes.

3.2.3 Características de los Usuarios

Nuestra aplicación va a tener dos tipos de usuarios:

Usuario: aquellas personas que desean conocer el estado de sus hijos e ingresan su DNI.

Administradores: son las personas encargadas de la gestión del autobús escolar, trabajadores del centro, como monitores del autobús, o personal de secretaría.

3.2.4 Restricciones

- Todas las personas que quieran acceder a la aplicación web van a necesitar cualquier dispositivo con acceso a Internet.
- Solo el usuario administrador va a poder crear modificar y eliminar registros de la aplicación.
- El lenguaje de programación que se utilizará será Node.js
- Se utilizarán plantillas Jade para la estructura de las vistas
- El *framework* para los estilos de las vistas será Semantic UI.
- Se trabajará con una base de datos MySQL.

3.2.5 Suposiciones y Dependencias

Se supone el correcto funcionamiento de la base de datos y del sistema donde se ejecute, así como la realización de copias de seguridad.

3.2.6 Requisitos Futuros

Como futuras mejoras al sistema, se pueden tener en cuenta:

- Chat directo entre parientes, alumnos y monitores a través de la aplicación.
- Registro único por persona en la aplicación web.
- Avisos automáticos a los parientes cuando el alumno entra y sale del autobús.

3.3 Requisitos Específicos

3.3.1 Interfaces Externas

3.3.1.1 Interfaces de usuarios

La interfaz de usuario constará de una barra de navegación superior, la cual incluirá un menú lateral con las acciones disponibles y un contenedor principal donde estará la información relacionada con el menú seleccionado.

3.3.1.2 Interfaces hardware

Los usuarios deben de disponer de un ordenador o un dispositivo que permita conexión a Internet para poder acceder a la aplicación web.

3.3.1.3 Interfaces software

La aplicación está desarrollada en Node.js, JQuery, Semantic UI y con una base de datos basada en MySQL, por lo que de este modo puede funcionar sobre cualquier navegador y sistema operativo.

3.3.1.4 Interfaces de comunicaciones

La comunicación entre el cliente y el servidor consiste en una comunicación de petición y respuesta, mediante el protocolo HTTP, y enviadas entre cliente/servidor con el protocolo TCP/IP.

3.3.2 Funciones

3.3.2.1 Usuario

- Búsqueda de alumno (Ilustración 7).
    - Introducción: Búsqueda de alumnos que hacen uso del autobús
    - Entrada: DNI y correo electrónico del usuario
    - Proceso: Búsqueda de alumno matriculado que coincida con los datos del pariente introducido.
    - Salida: Tabla de estado del alumno.

- Listado de hijos (Ilustración 7).

    - Introducción: Listado de hijos del pariente que ha realizado la consulta
    - Entrada: Datos del pariente
    - Proceso: Se realiza una búsqueda con los registros coincidentes.
    - Salida: Se muestra el resultado en una tabla con los datos del alumno.

- Iniciar sesión (Ilustración 8)

    - Introducción: Inicio de sesión del usuario con la finalidad de gestionar autobuses.
    - Entrada: Usuario y contraseña del administrador.
    - Proceso: Se validan los datos introducidos por el usuario y se confirma o se deniega la entrada a la aplicación.
    - Salida: Se visualiza la página inicial de gestión.

3.3.2.2 Administrador

- Crear autobuses (Ilustración 9)

    - Introducción: Crear autobús nuevo.
    - Entrada: Datos del autobús, matrícula, ruta, capacidad, conductor y monitor.
    - Proceso: Añade un nuevo registro a la base de datos de la tabla autobús.
    - Salida: Confirmación de creación correcta del autobús.

- Listar autobuses (Ilustración 9)

    - Introducción: Listado de autobuses disponibles en el centro.
    - Entrada: -
    - Proceso: Recupera de la base de datos todos los autobuses registrados.
    - Salida: Muestra una tabla con todos los autobuses y atributos respectivos.

- Filtrar autobuses (Ilustración 9)

    - Introducción: Permite realizar un filtro en la tabla de autobuses.
    - Entrada: Matrícula, ruta, monitor, conductor.
    - Proceso: Recupera de la base de datos todos los autobuses que coincidan con el filtro.
    - Salida: Muestra una tabla con los autobuses filtrados.

- Modificar autobuses (Ilustración 9)

    - Introducción: Modifica los datos del autobús.
    - Entrada: Matrícula, ruta, capacidad, conductor y monitor.
    - Proceso: Modifica el registro en la base de datos del autobús con los datos introducidos.
    - Salida: Retorno al listado de autobuses con el autobús modificado.

- Eliminar autobuses (Ilustración 9)

    - Introducción: Elimina el autobús seleccionado.
    - Entrada: Se pasa la id por contexto al pulsar el botón “Eliminar” del autobús correspondiente.
    - Proceso: Borra el registro del autobús de la base de datos.
    - Salida: -

- Crear conductores (Ilustración 10)

    - Introducción: Creación de conductor nuevo.
    - Entrada: Nombre, teléfono.
    - Proceso: Añade un nuevo registro a la base de datos de la tabla conductor.
    - Salida: Confirmación de conductor creado correctamente.

- Listar conductores (Ilustración 10)

    - Introducción: Listado de conductores del centro.
    - Entrada: -
    - Proceso: Recupera de la base de datos todos los conductores registrados.
    - Salida: Muestra una tabla con todos los conductores y atributos respectivos.

- Filtrar conductores (Ilustración 10)

  - Introducción: Permite realizar un filtro en la tabla de conductores.
  - Entrada: Nombre.
  - Proceso: Recupera de la base de datos todos los conductores que coincidan con el filtro.
  - Salida: Muestra una tabla con los conductores filtrados.

- Modificar conductores (Ilustración 10)

  - Introducción: Modifica los datos del conductor.
  - Entrada: Nombre, teléfono.
  - Proceso: Modifica el registro en la base de datos del conductor con los datos introducidos.
  - Salida: Retorno al listado de conductores con el conductor modificado.

- Eliminar conductores (Ilustración 10)

  - Introducción: Elimina al conductor seleccionado.
  - Entrada: Se pasa la id por contexto al pulsar el botón “Eliminar” del conductor correspondiente.
  - Proceso: Borra el registro del conductor de la base de datos.
  - Salida: -

- Crear monitores (Ilustración 11)

  - Introducción: Creación de monitor nuevo.
  - Entrada: Nombre, email, teléfono.
  - Proceso: Añade un nuevo registro a la base de datos de la tabla conductor.
  - Salida: Confirmación de monitor creado correctamente.

- Listar monitores (Ilustración 11)

  - Introducción: Listado de monitores del centro.
  - Entrada: -
  - Proceso: Recupera de la base de datos todos los monitores registrados.
  - Salida: Muestra una tabla con todos los monitores y atributos respectivos.

- Filtrar monitores (Ilustración 11)

    - Introducción: Permite realizar un filtro en la tabla de monitores.
    - Entrada: Nombre, correo.
    - Proceso: Recupera de la base de datos todos los monitores que coincidan con el filtro.
    - Salida: Muestra una tabla con los monitores filtrados.

- Modificar monitores (Ilustración 11)

    - Introducción: Modifica los datos del monitor seleccionado.
    - Entrada: Nombre, email, teléfono.
    - Proceso: Modifica el registro en la base de datos del monitor con los datos introducidos.
    - Salida: Retorno al listado de monitores con el monitor modificado.

- Eliminar monitores (Ilustración 11)

    - Introducción: Elimina al monitor seleccionado
    - Entrada: Se pasa la id por contexto al pulsar el botón “Eliminar” del monitor correspondiente.
    - Proceso: Borra el registro del monitor de la base de datos.
    - Salida: -

- Crear rutas (Ilustración 12)

    - Introducción: Creación de ruta nueva.
    - Entrada: Nombre, paradas, ruta de ida, ruta de vuelta.
    - Proceso: Añade un nuevo registro a la base de datos de la tabla ruta.
    - Salida: Confirmación de ruta creada correctamente.

- Listar rutas (Ilustración 12).

    - Introducción: Listado de rutas existentes realizadas por los autobuses.
    - Entrada: -
    - Proceso: Recupera de la base de datos todas las rutas registradas.
    - Salida: Muestra una tabla con todas las rutas y atributos respectivos.

- Filtrar rutas (Ilustración 12).

    - Introducción: Permite realizar un filtro en la tabla de rutas.
    - Entrada: Nombre.
    - Proceso: Recupera de la base de datos todas las rutas que coincidan con el filtro.
    - Salida: Muestra una tabla con las rutas filtradas.

- Modificar rutas (Ilustración 12).

    - Introducción: Modifica los datos de la ruta seleccionada.
    - Entrada: Nombre, paradas, ruta de ida, ruta de vuelta.
    - Proceso: Modifica el registro en la base de datos de la ruta con los datos introducidos.
    - Salida: Retorno al listado de rutas con la ruta modificada.

- Eliminar rutas (Ilustración 12).

    - Introducción: Elimina la ruta seleccionada
    - Entrada: Se pasa la id de la ruta por contexto al pulsar el botón “Eliminar” de la ruta correspondiente.
    - Proceso: Borra el registro de la ruta de la base de datos.
    - Salida: -

- Enviar correo (Ilustración 12).

    - Introducción: Se envía un correo desde la ruta seleccionada para informar de la existencia de un atasco a los parientes que tengan hijos viajando en esa ruta.
    - Entrada: Se pasa la id de la ruta por contexto al enviar el correo.
    - Proceso: A partir de la id de la ruta, se buscan a los alumnos que estén viajando en ella y se envía un correo electrónico automatizado a los parientes.
    - Salida: Correo de salida avisando de atasco en la ruta en la que se encuentra el alumno.

- Crear rutas de ida (Ilustración 14).

    - Introducción: Creación de rutas de ida.
    - Entrada: Nombre, paradas de ida.
    - Proceso: Añade un nuevo registro a la base de datos de la tabla rutaIda.
    - Salida: Confirmación de ruta de ida creada correctamente.

- Listar rutas de ida (Ilustración 14).

    - Introducción: Listado de rutas de ida existentes.
    - Entrada: -
    - Proceso: Recupera de la base de datos todas las rutas de ida registradas.
    - Salida: Muestra una tabla con todas las rutas de ida y atributos respectivos.

- Filtrar rutas de ida (Ilustración 14).

    - Introducción: Permite realizar un filtro en la tabla de rutas de ida.
    - Entrada: Nombre.
    - Proceso: Recupera de la base de datos todas las rutas de ida que coincidan con el filtro.
    - Salida: Muestra una tabla con las rutas de ida filtradas.

- Modificar rutas de ida (Ilustración 14).

    - Introducción: Modifica los datos de la ruta de ida seleccionada.
    - Entrada: Nombre, paradas ida.
    - Proceso: Modifica el registro en la base de datos de la ruta de ida con los datos introducidos.
    - Salida: Retorno al listado de rutas de ida con la ruta de ida modificada

- Eliminar rutas de ida (Ilustración 14).

    - Introducción: Elimina la ruta de ida seleccionada
    - Entrada: Se pasa la id de la ruta de ida por contexto al pulsar el botón “Eliminar”.
    - Proceso: Borra el registro de la ruta de ida de la base de datos.
    - Salida: -

- Crear rutas de vuelta (Ilustración 15).

    - Introducción: Creación de rutas de vuelta.
    - Entrada: Nombre, paradas de vuelta.
    - Proceso: Añade un nuevo registro a la base de datos de la tabla rutaVuelta.
    - Salida: Confirmación de ruta de vuelta creada correctamente.

- Listar rutas de vuelta (Ilustración 15).

    - Introducción: Listado de rutas de vuelta existentes.
    - Entrada: -
    - Proceso: Recupera de la base de datos todas las rutas de vuelta registradas.
    - Salida: Muestra una tabla con todas las rutas de vuelta y atributos respectivos.

- Filtrar rutas de vuelta (Ilustración 15).

    - Introducción: Permite realizar un filtro en la tabla de rutas de vuelta.
    - Entrada: Nombre.
    - Proceso: Recupera de la base de datos toaos las rutas de vuelta que coincidan con el filtro.
    - Salida: Muestra una tabla con las rutas de vuelta filtradas.

- Modificar rutas de vuelta (Ilustración 15).

    - Introducción: Modifica los datos de la ruta de vuelta seleccionada.
    - Entrada: Nombre, paradas vuelta.
    - Proceso: Modifica el registro en la base de datos de la ruta de vuelta con los datos introducidos.
    - Salida: Retorno al listado de rutas de vuelta con la ruta de vuelta modificada

- Eliminar rutas de vuelta (Ilustración 15).

    - Introducción: Elimina la ruta de vuelta seleccionada
    - Entrada: Se pasa la id de la ruta de vuelta por contexto al pulsar el botón “Eliminar”.
    - Proceso: Borra el registro de la ruta de vuelta de la base de datos.
    - Salida: -

- Crear matriculados (Ilustración 13).

    - Introducción: Creación de alumnos matriculados.
    - Entrada: Nombre, nombre 2º, apellido paterno, apellido materno, DNI, ruta, curso escolar, estado, ruta ida, ruta vuelta.
    - Proceso: Añade un nuevo registro a la base de datos de la tabla matriculados.
    - Salida: Confirmación de alumno creado correctamente.

- Listar matriculados (Ilustración 13).

    - Introducción: Listado de los alumnos matriculados en el centro.
    - Entrada: -
    - Proceso: Recupera de la base de datos todos los alumnos registrados.
    - Salida: Muestra una tabla con todos los alumnos matriculados y atributos respectivos.

- Filtrar matriculados (Ilustración 13).

    - Introducción: Permite realizar un filtro en la tabla de matriculados.
    - Entrada: DNI.
    - Proceso: Recupera de la base de datos todos los matriculados que coincidan con el filtro.
    - Salida: Muestra una tabla con los matriculados filtrados.

- Modificar matriculados (Ilustración 13).

    - Introducción: Modifica los datos de los alumnos.
    - Entrada: Nombre, nombre 2º, apellido paterno, apellido materno, DNI, ruta, curso escolar, estado, ruta ida, ruta vuelta.
    - Proceso: Modifica el registro en la base de datos del alumno con los datos introducidos.
    - Salida: Retorno al listado de matriculados con el alumno modificado.

- Eliminar matriculados (Ilustración 13).

    - Introducción: Elimina al alumno seleccionado.
    - Entrada: Se pasa la id del alumno por contexto al pulsar el botón “Eliminar”.
    - Proceso: Borra el registro del alumno de la base de datos.
    - Salida: -

- Notificación de estado del matriculado (Ilustración 13).

  - Introducción: Envío de mensaje informativo a los parientes por Telegram del estado del matriculado.
  - Entrada: -
  - Proceso: Se envía una notificación *push* al móvil de los parientes teniendo en cuenta el estado guardado en la base de datos del matriculado.
  - Salida: Mensaje enviado por Telegram del estado del alumno.

- Crear cursos escolares (Ilustración 16).

  - Introducción: Crea un curso escolar nuevo.
  - Entrada: Curso, clase.
  - Proceso: Añade un nuevo registro a la tabla curso escolar de la base de datos.
  - Salida: Confirmación de curso escolar creado correctamente.

- Listar cursos escolares (Ilustración 16).

  - Introducción: Listado de cursos escolares del centro.
  - Entrada: -
  - Proceso: Recupera de la base de datos todos los cursos escolares registrados.
  - Salida: Muestra una tabla con todos los cursos escolares y atributos respectivos.

- Filtrar cursos escolares (Ilustración 16).

  - Introducción: Permite realizar un filtro en la tabla de cursos escolares.
  - Entrada: Curso.
  - Proceso: Recupera de la base de datos todos los cursos escolares que coincidan con el filtro.
  - Salida: Muestra una tabla con los cursos escolares filtrados.

- Modificar cursos escolares (Ilustración 16).

  - Introducción: Modifica el curso escolar seleccionado.
  - Entrada: Curso, clase.
  - Proceso: Modifica el registro en la tabla curso escolar de la base de datos con los datos introducidos.
  - Salida: Retorno a la lista de cursos escolares con el curso escolar modificado.

- Eliminar cursos escolares (Ilustración 16).

    - Introducción: Elimina el curso escolar seleccionado.
    - Entrada: Se pasa la id del curso escolar al pulsar el botón “Eliminar”.
    - Proceso: Borra el registro de la base de datos.
    - Salida: -

- Crear parientes (Ilustración 17).

    - Introducción: Crea un nuevo pariente.
    - Entrada: Nombre, foto, hijo/a, email, teléfono.
    - Proceso: Añade un nuevo registro a la tabla pariente de la base de datos.
    - Salida: Confirmación de pariente creado correctamente.

- Listar parientes (Ilustración 17).

    - Introducción: Listado de parientes.
    - Entrada: -
    - Proceso: Recupera de la base de datos todos los parientes registrados.
    - Salida: Muestra una tabla con todos los parientes y atributos respectivos.

- Filtrar parientes (Ilustración 17).

    - Introducción: Permite realizar un filtro en la tabla de parientes.
    - Entrada: Nombre, hijo, correo.
    - Proceso: Recupera de la base de datos todos los parientes que coincidan con el filtro.
    - Salida: Muestra una tabla con los parientes filtrados.

- Modificar parientes (Ilustración 17).

    - Introducción: Modifica el pariente seleccionado.
    - Entrada: Nombre, foto, hijo/a, email, teléfono.
    - Proceso: Modifica el registro en la tabla pariente de la base de datos con los datos introducidos.
    - Salida: Retorno a la lista de parientes con el pariente modificado.

- Eliminar parientes (Ilustración 17).

    - Introducción: Elimina al pariente seleccionado.
    - Entrada: Se pasa la id del pariente por contexto al pulsar el botón “Eliminar”.
    - Proceso: Borra el registro de la tabla pariente en la base de datos.
    - Salida: -

- Subir foto (Ilustración 17).

    - Introducción: Al crear o modificar un pariente se subirá una foto, la cual se quedará guardada en una carpeta estática.
    - Entrada: Foto en formato PNG o JPG.
    - Proceso: Una vez elegida la foto, se va a guardar en la base de datos el nombre del fichero y en una carpeta estática el contenido de la imagen, para a continuación ser mostrada en el listado de parientes
    - Salida: Imagen seleccionada.

- Crear paradas (Ilustración 18).

    - Introducción: Crea una parada nueva.
    - Entrada: Nombre, ruta, ruta de ida, ruta de vuelta, tiempo, longitud, latitud, hora ida de llegada, hora ida de salida, hora vuelta de llegada, hora vuelta de salida.
    - Proceso: Recupera de la base de datos todos los conductores registrados.
    - Salida: Confirmación de parada creada correctamente.

- Listar paradas (Ilustración 18).

    - Introducción: Listado de paradas.
    - Entrada: -
    - Proceso: Recupera de la base de datos todas las paradas registradas.
    - Salida: Muestra una tabla con todas las paradas y atributos respectivos.

- Filtrar paradas (Ilustración 18).

    - Introducción: Permite realizar un filtro en la tabla de paradas.
    - Entrada: Nombre, ruta
    - Proceso: Recupera de la base de datos todas las paradas que coincidan con el filtro.
    - Salida: Muestra una tabla con las paradas filtradas.

- Modificar paradas (Ilustración 18).

    - Introducción: Modifica la parada seleccionada con nuevos datos.
    - Entrada: Nombre, ruta, ruta de ida, ruta de vuelta, tiempo, longitud, latitud, hora ida de llegada, hora ida de salida, hora vuelta de llegada, hora vuelta de salida.
    - Proceso: Modifica el registro en la tabla paradas de la base de datos con los datos introducidos.
    - Salida: Retorno a la lista de paradas con la parada modificada.

- Eliminar paradas (Ilustración 18).

    - Introducción: Elimina la parada seleccionada.
    - Entrada: Se pasa la id de la parada por contexto al pulsar el botón “Eliminar”.
    - Proceso: Borra el registro de la tabla parada en la base de datos
    - Salida: -

3.3.3 Requisitos de Rendimiento

La respuesta dada por la página web va a ser en tiempo real.

La respuesta de la base de datos va a ser de milisegundos, por lo que vía web no va a haber ningún retardo en la muestra y procesamiento de datos. Para que hubiera retardo, la base de datos debería de tener más de cien mil registros.

3.3.4 Restricciones de Diseño

El diseño va a variar en función del dispositivo que se esté usando.

Se ha realizado un diseño general para cualquier tipo de situación.

3.3.5 Atributos del Sistema

3.3.5.1 Integridad

Para poder guardar los datos, se ha utilizado una base de datos MySQL, permitiendo la persistencia de datos y con la posibilidad de realizar *backups*[3]

[3] Backup: Proceso realizado cuando se quiere guardar una copia de la base de datos.

3.3.5.2 Mantenimiento

El mantenimiento debe de ser llevado por el administrador de sistema, al cual se le facilitaría acceso a la máquina donde se encuentra instalada la aplicación.

3.3.5.3 Seguridad

Se dispone de un inicio de sesión para distinguir administradores de usuarios normales. De este modo solo el administrador podrá realizar la parte de gestión, y el usuario solo podrá consultar sus datos.

3.3.6 Otros Requisitos

3.3.6.1 Base de datos

La aplicación hace uso de una base de datos MySQL, donde se va a almacenar toda la información de los autobuses, matriculados, rutas, paradas, conductores, monitores y parientes.

4. Diseño de la aplicación

En este apartado se va a describir la estructura del proyecto mediante diagramas y casos de uso y se va a analizar la aplicación a desarrollar para poder entender su funcionalidad de manera más sencilla. Al finalizar esta sección, vamos a obtener un modelo con todos los actores que van a interactuar con los objetos del sistema a través de las diferentes relaciones y acciones.

4.1 Diagrama de clases

Un diagrama de clases describe la estructura del sistema [1] que se va a desarrollar, nos muestra sus clases, atributos y relaciones entre ellos, ofreciendo información del funcionamiento del sistema.

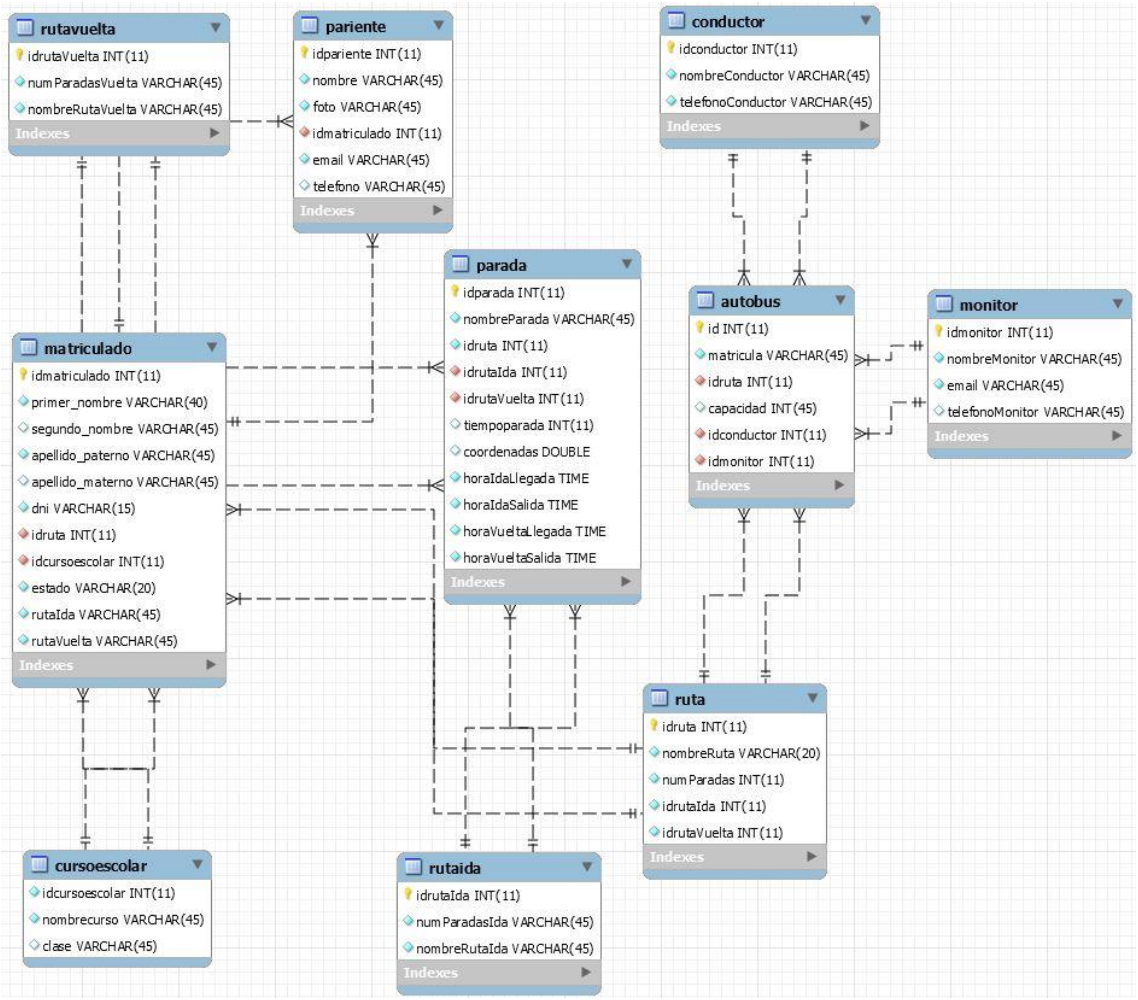


Ilustración 5: Diagrama de clases

De la Ilustración 5 se puede extraer la siguiente información:

Un **autobús** tiene como datos, su identificador primario, su matrícula, el identificador de la ruta que se le ha asignado, su capacidad, el identificador del conductor que se le ha asignado, y el identificador del monitor asignado. Sólo puede tener un conductor, un monitor y una ruta asignada al mismo tiempo.

El **conductor** del **autobús** tiene como datos personales su identificador primario, nombre y su teléfono. No está limitado al mismo autobús, por lo que puede conducir cualquier autobús que tenga disponible el centro.

El **monitor** del **autobús** tiene como datos personales su identificador primario, nombre, email y teléfono. No tiene como restricción monitorizar el mismo autobús, por lo que puede ser asignado a cualquier otro en cualquier momento.

La **ruta** que sigue el **autobús** tiene como datos su identificador primario, nombre, el número de paradas, el identificador de la ruta de ida asignada y el identificador de la ruta de vuelta asignada. La ruta puede ser asignada a varios autobuses y puede ser asignada a muchos alumnos.

Los **matriculados** tienen como datos personales el identificador primario, primer nombre, el segundo nombre, el apellido paterno, el apellido materno, el DNI, el identificador de la ruta asignada, el identificador del curso escolar asignado, su estado, el identificador de la ruta de ida asignada y el identificador de la ruta de vuelta asignada. Por lo tanto, un alumno matriculado va a tener varios parientes, una ruta de ida, una ruta de vuelta, una ruta y un curso escolar.

El **curso escolar** tiene como datos el identificador primario, el nombre del curso y la clase. Un curso puede tener más de un alumno matriculado.

Tanto la **ruta de ida** como la **ruta de vuelta** van a tener como datos, su identificador personal, su nombre y su número de paradas. Van a poder estar asociadas con diferentes rutas y paradas, y van a tener asociado a un alumno.

Los **parientes** van a tener como datos personales un identificador primario, nombre, foto, identificador del matriculado asignado, email y teléfono. Van a estar asociados con los alumnos matriculados, los cuáles serán sus hijos/hijas.

Se ha utilizado el programa MySQL Workbench (Ilustración 6) para la creación de este diagrama, ya que nos ha permitido utilizar la opción de ingeniería inversa, en la cual al pasarle una base de datos nos genera un diagrama de clases [16].

Ilustración 6: MySQL Workbench

4.2 Casos de uso

Partiendo de los requisitos funcionales descritos en el apartado 3.3.2 podemos hacer los siguientes casos de uso.

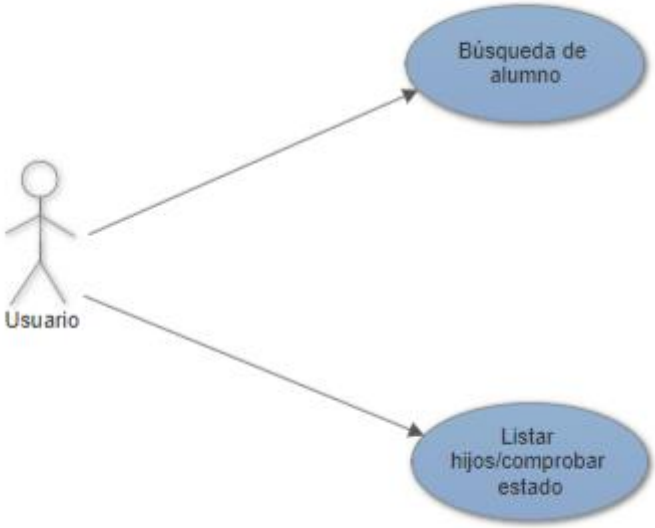


Ilustración 7: Casos de uso – Usuario

Ilustración 8: Casos de uso - Usuario - Iniciar Sesión

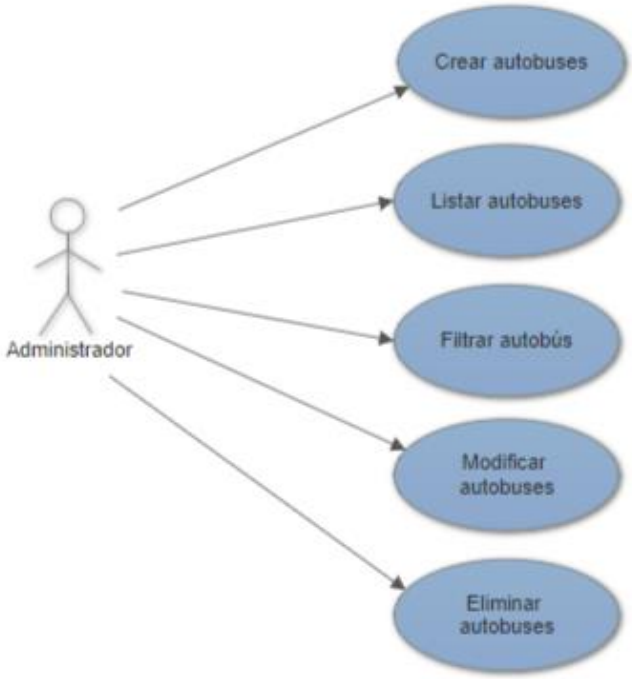


Ilustración 9: Casos de uso - Administrador – Autobuses

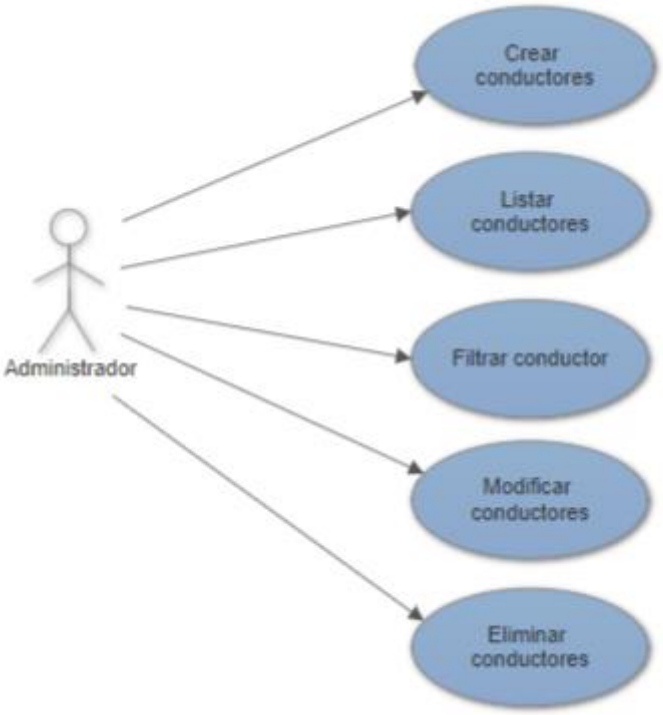


Ilustración 10: Casos de uso - Administrador - Conductores

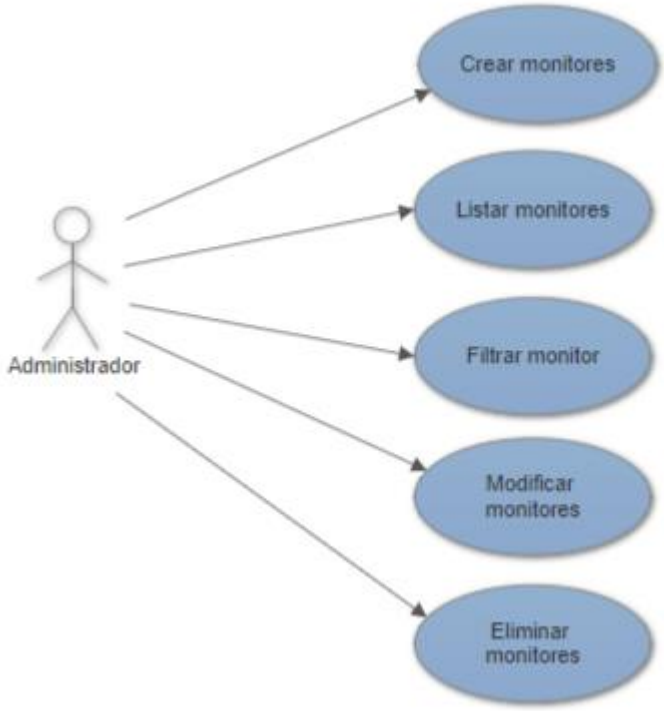


Ilustración 11: Casos de uso - Administrador - Monitores

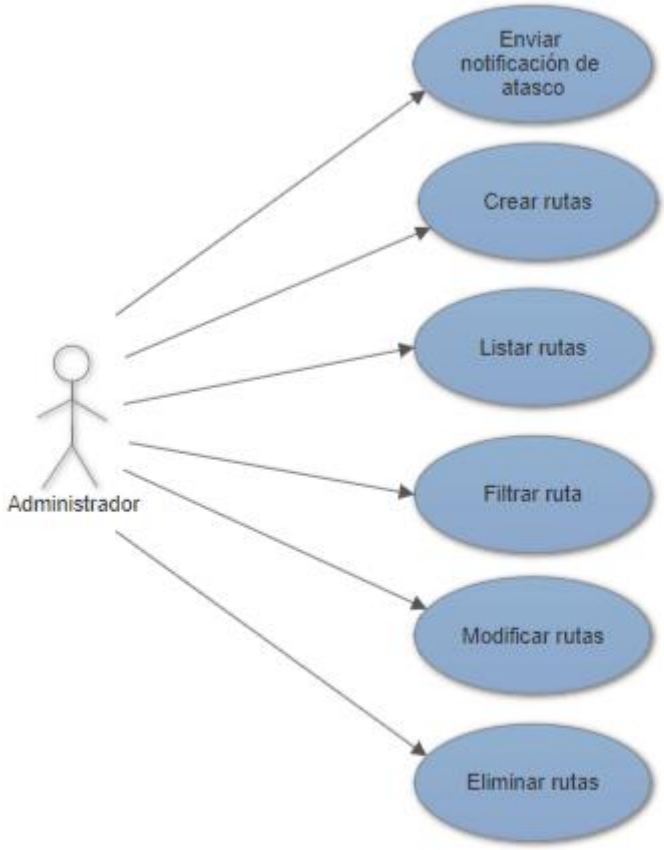


Ilustración 12: Casos de uso - Administrador - Rutas

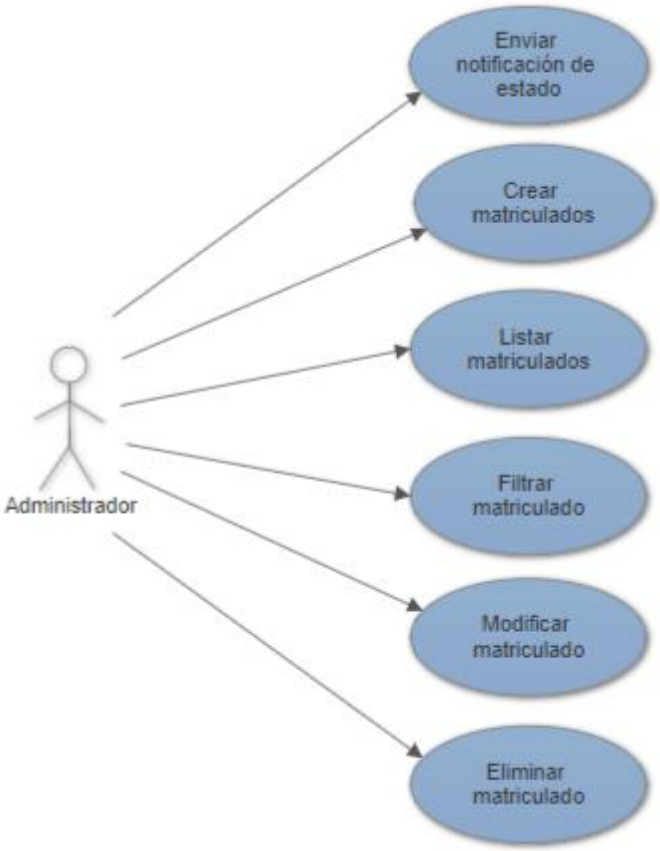


Ilustración 13: Casos de uso - Administrador – Matriculados

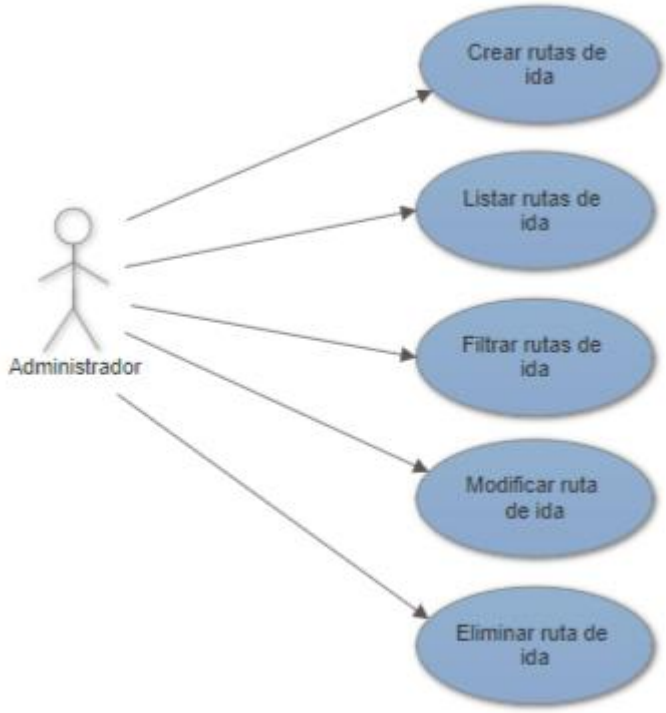


Ilustración 14: Casos de uso - Administrador - Rutas de ida

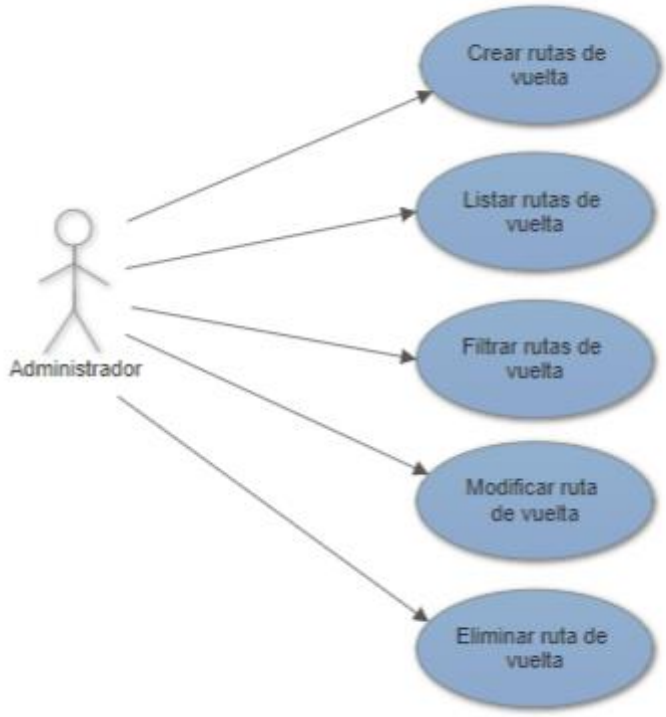


Ilustración 15: Casos de uso - Administrador - Rutas de vuelta

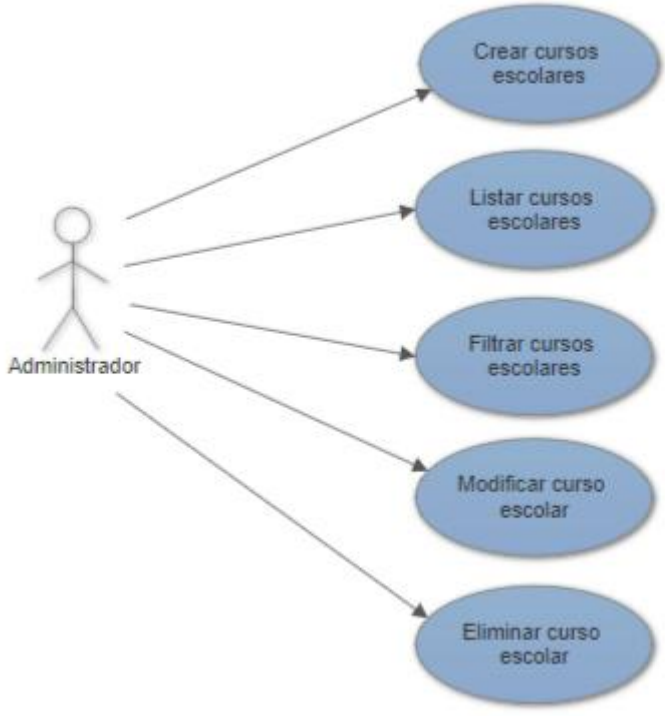


Ilustración 16: Casos de uso - Administrador - Cursos escolares

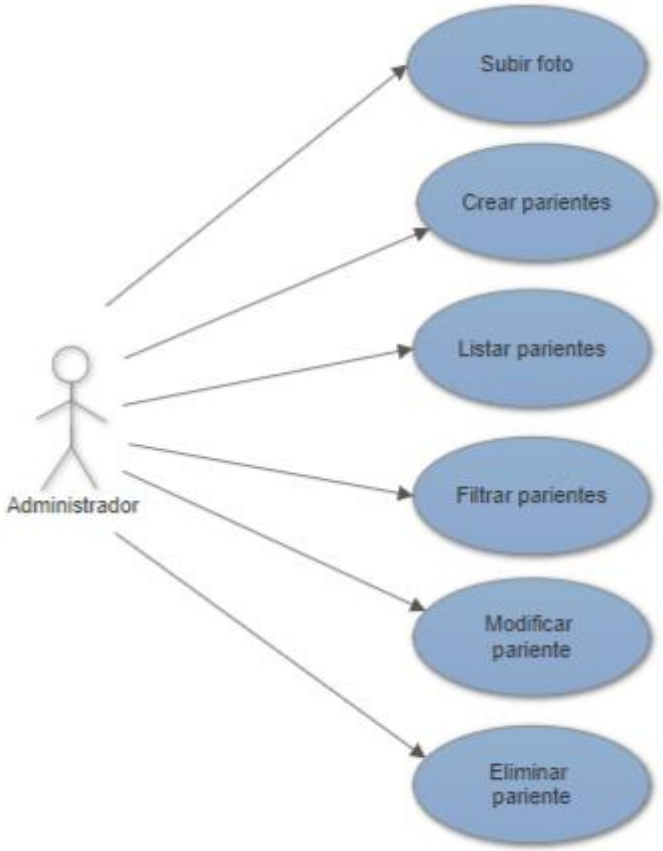


Ilustración 17: Casos de uso - Administrador - Parientes

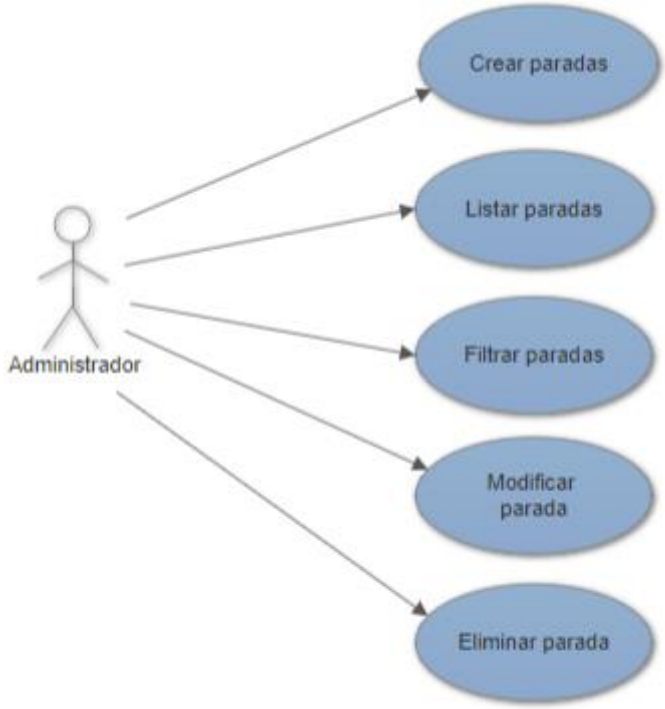


Ilustración 18: Casos de uso - Administrador - Paradas

Como se puede observar, el usuario solamente va a poder buscar a su hijo para comprobar su estado.

El personal que busque administrar la aplicación web deberá de iniciar la sesión. De este modo se le considerará administrador y tendrá acceso a la gestión de todos los objetos que incluye el proyecto.

4.3 Prototipo y versión final

En esta sección se van a presentar los diferentes prototipos realizados, así como las versiones finales del diseño de la aplicación, donde van a aparecer todas las vistas que la componen.

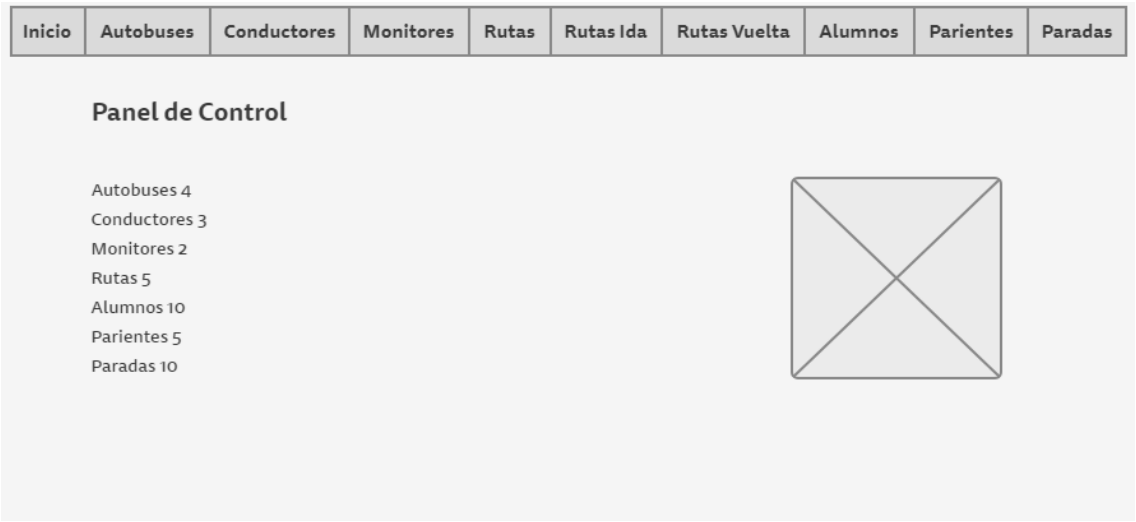


Ilustración 19: Vista - Prototipo inicio

En la Ilustración 19 se muestra la página de inicio de gestión de la aplicación. Con un menú superior donde se encuentran todas las opciones de gestión disponibles.

Como se verá más adelante, en la versión final, el menú superior pasó a ser un menú lateral desplegable, proporcionando una página inicial más limpia y un menú *responsive* para el diseño móvil.

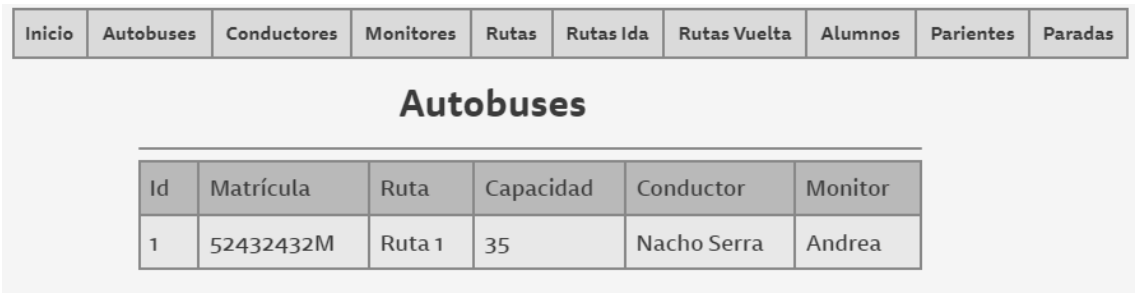
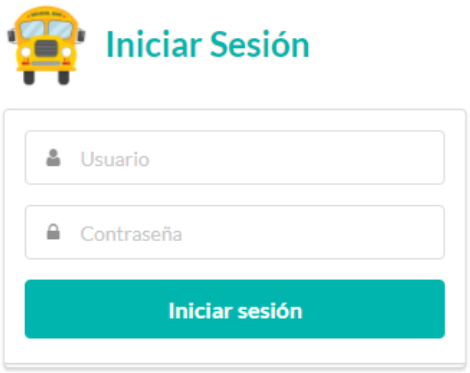


Ilustración 20: Vista - Prototipo Listar autobuses

La versión prototipo del listado de autobuses (Ilustración 20) fue mejorada en la versión final con una nueva columna de acciones disponibles y dos botones con nueva funcionalidad.

Ilustración 21: Vista - Prototipo Crear autobús

La Ilustración 21 enseña cómo iba a ser la creación de un autobús. En la versión final se añadieron etiquetas a cada entrada de datos, así como campos seleccionables para facilitar la creación.

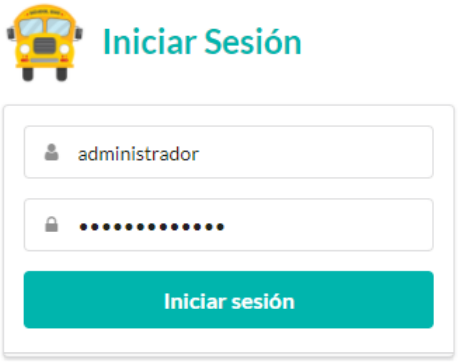


La primera vista de nuestra aplicación es el Inicio de sesión. Para poder realizar cualquier acción de gestión va a ser necesario utilizar el usuario administrador y la contraseña administrador (Ilustración 22).

Ilustración 22: Vista - Iniciar sesión

Si el usuario o la contraseña están en blanco o son incorrectos, va a aparecer un mensaje informativo en rojo avisando del problema (Ilustración 23).

Ilustración 23: Vista - Validación iniciar sesión



Una vez se ha introducido el usuario y contraseña de manera correcta, podremos iniciar sesión y empezar a gestionar los autobuses y todas sus relaciones (Ilustración 24).

Ilustración 24: Vista – Inicio de sesión correcto

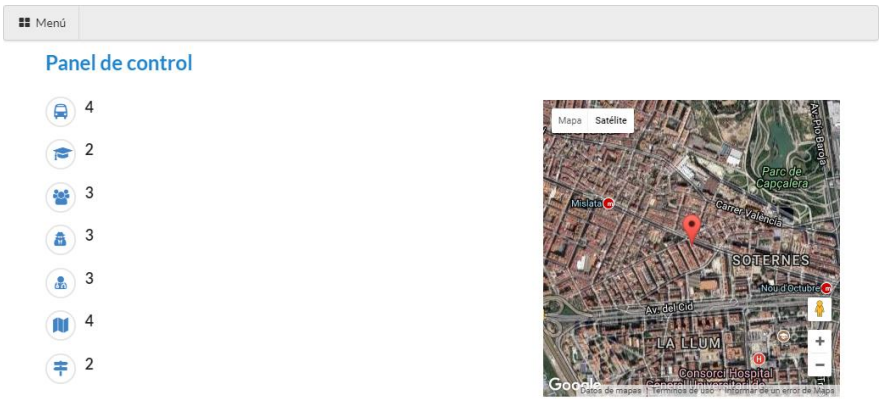


Ilustración 25: Vista - Inicio

Al iniciar la sesión se va a mostrar la vista del panel de control (Ilustración 25). En la parte izquierda tenemos un menú desplegable lateral donde van a aparecer las diversas gestiones a realizar. En el centro, una serie de iconos de acceso rápido para gestionar autobuses, matriculados, parientes, conductores, monitores rutas y paradas, así como la cantidad actual de cada uno de los objetos.

En la parte derecha, un mapa de *Google Maps* para consultar la ruta y realizar marcas.

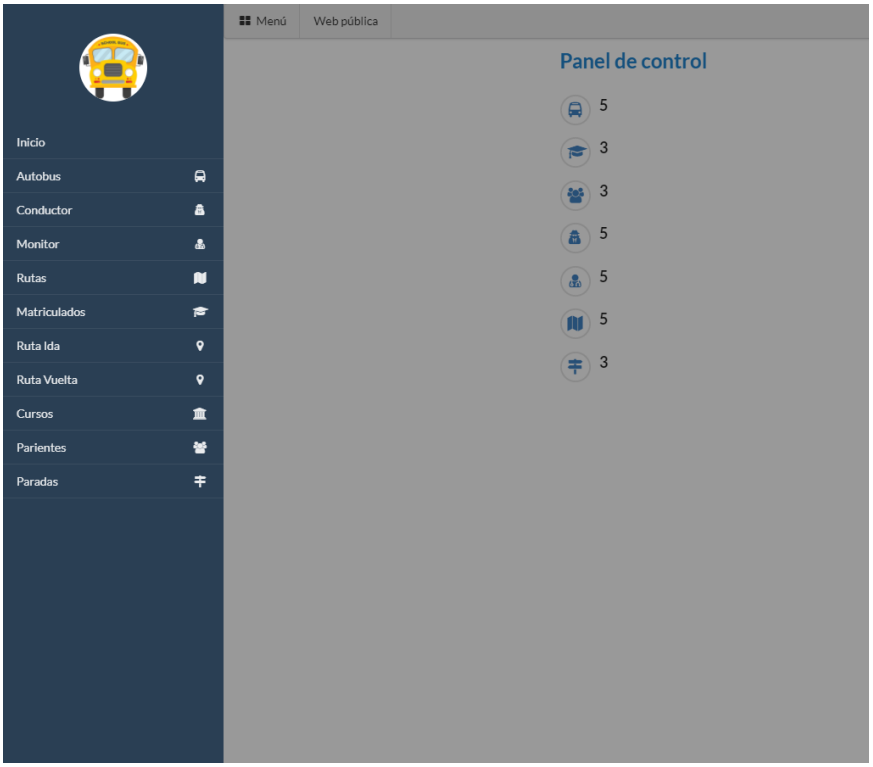


Ilustración 26: Vista - Inicio, menú desplegable

Al hacer *click* en el botón de "Menú", se muestra un menú desplegable lateral (Ilustración 26) en el cual vamos a tener acceso a todo el apartado de gestión de la aplicación. Está organizado por las categorías de los objetos más importantes en la administración de autobuses escolares.

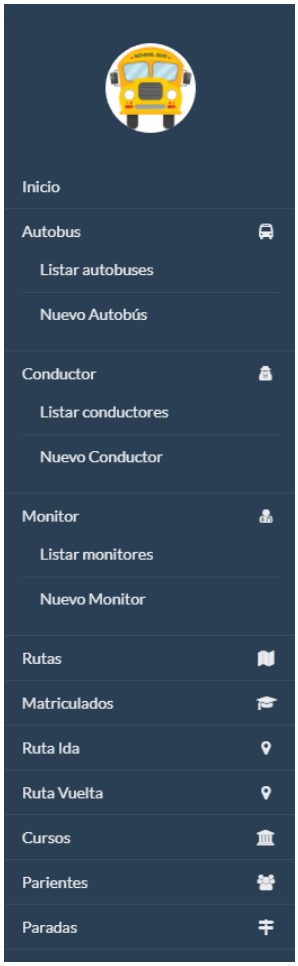


Ilustración 27: Vista - Menú lateral desplegado

En cada objeto del menú desplegable (Ilustración 27) vamos a tener referencias a las diferentes vistas que componen la aplicación:

- **Listar**: Va a contener una nueva vista formada por una tabla con todos los objetos que existan de esa categoría, con todos sus atributos en cada columna.

- **Nuevo**: Vista de creación del nuevo objeto. Una vez introducidos los datos se creará el nuevo registro en la base de datos.

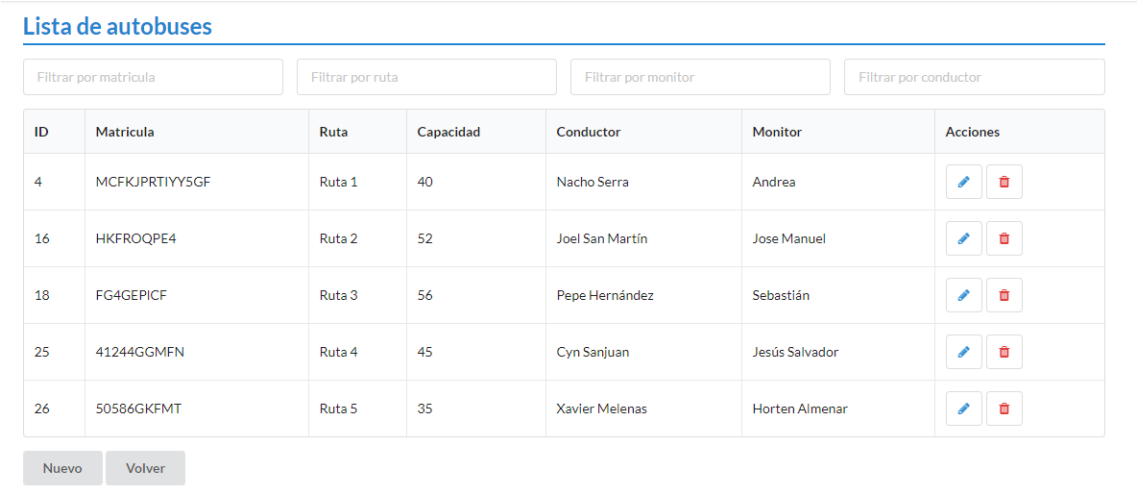


Ilustración 28: Vista - Listar autobuses

En la Ilustración 28 podemos observar diferentes filtros disponibles para acotar búsquedas, una tabla con diferentes columnas, que pertenecen a los atributos del objeto y a las acciones que se pueden realizar con cada registro(autobús).

En las acciones existen dos botones, uno para modificar el registro o para eliminarlo. Este diseño se va a mantener en las diferentes vistas.

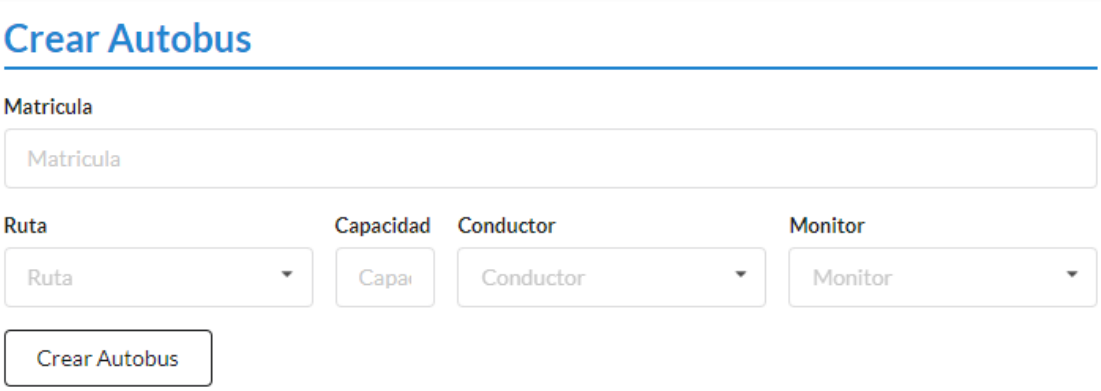


Ilustración 29: Vista - Crear autobús

En la vista de creación (Ilustración 29), se rellenarán los campos con los atributos que se solicitan. Se destaca que, en el ejemplo, los campos “Ruta”, “Conductor” y “Monitor” son campos seleccionables. Al hacer *click* en ellos aparecerán todos los objetos creados y se relacionarán con el autobús.

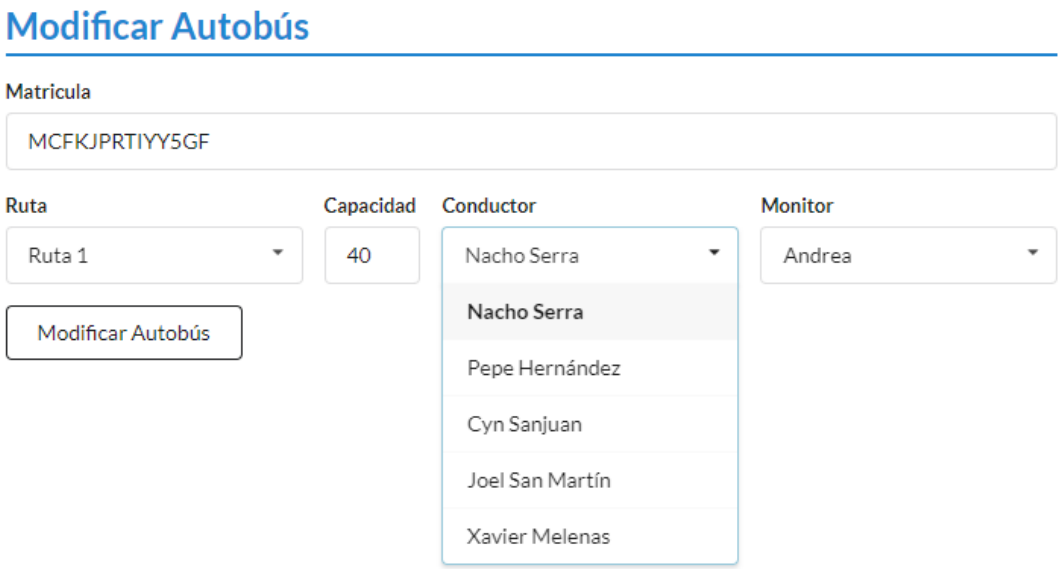


Ilustración 30: Vista - Modificar Autobús

La ilustración 30 presenta la vista de modificación del objeto, en este caso del autobús.

Esta vista retiene los atributos con los que se había creado el autobús, y permite realizar correcciones y modificaciones.

Una vez se realicen los cambios pertinentes, al hacer click en “Modificar Autobús” volveremos al listado de autobuses con los cambios guardados correctamente tanto en la base de datos, como en la vista del listado.

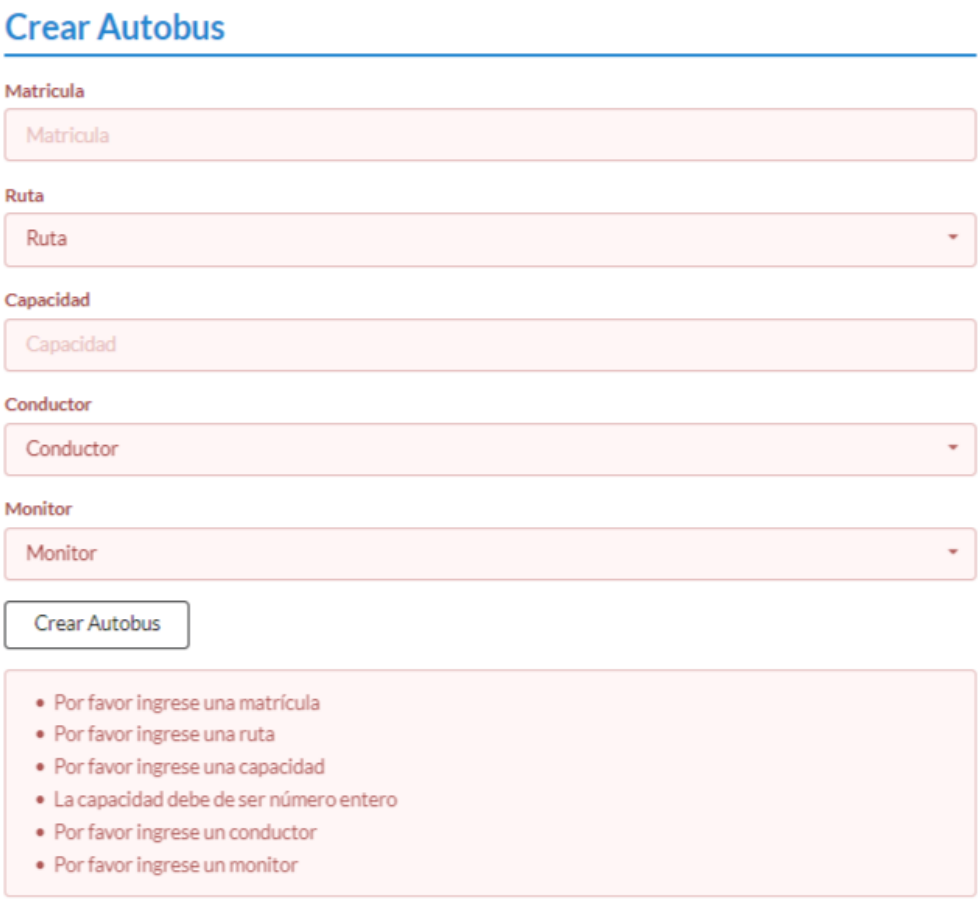


Ilustración 31: Vista – Validación

Como se ve en la Ilustración 31, se han realizado validaciones para que no se permita introducir datos erróneos. Los campos que se validan son:

- Números enteros: no se admitirán decimales, solo enteros
- Números decimales: solo se admitirán números decimales.
- Correos electrónicos: el correo electrónico que se introduzca debe de ser valido
- Campos vacíos: si el campo está vacío no se podrá entregar el formulario.

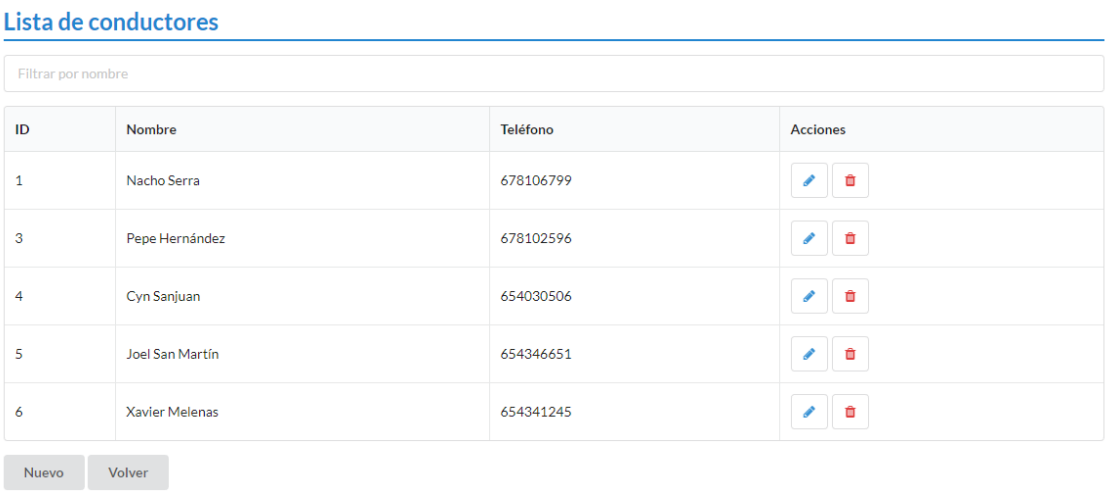


Ilustración 32: Vista - Listar conductores

En la vista de conductores (Ilustración 32), estarán todos aquellos conductores disponibles del centro que vayan a tener un autobús asignado. Las acciones disponibles van a ser filtrar conductor por nombre, crear un conductor nuevo, modificar los conductores existentes, eliminar conductores o volver a la página anterior.

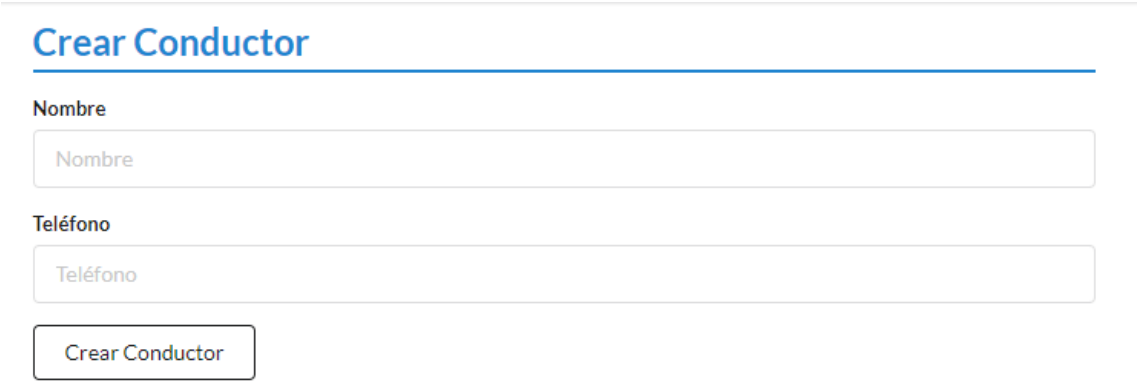


Ilustración 33: Vista - Crear conductor

Los datos a introducir en la creación del conductor (Ilustración 33) van a ser el nombre y el teléfono.

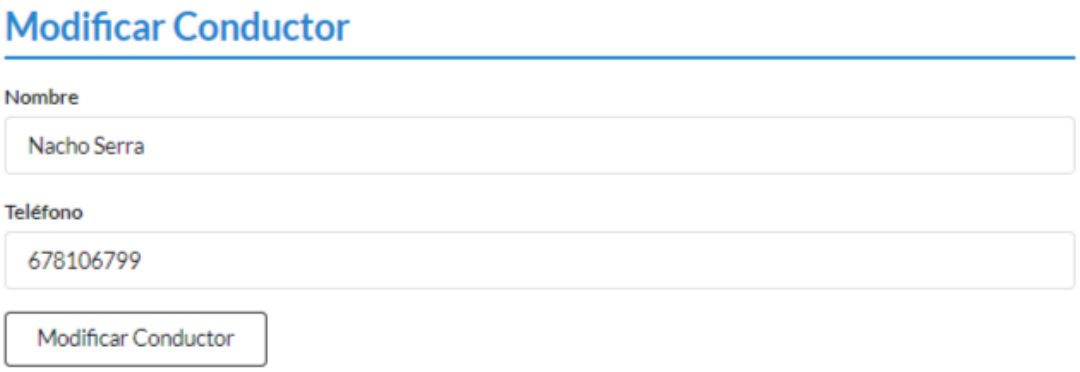


Ilustración 34: Vista - Modificar conductor

Al hacer uso de la acción "Modificar", se van a mantener los datos existentes (Ilustración 34) y se va a permitir modificar cualquiera de ellos.

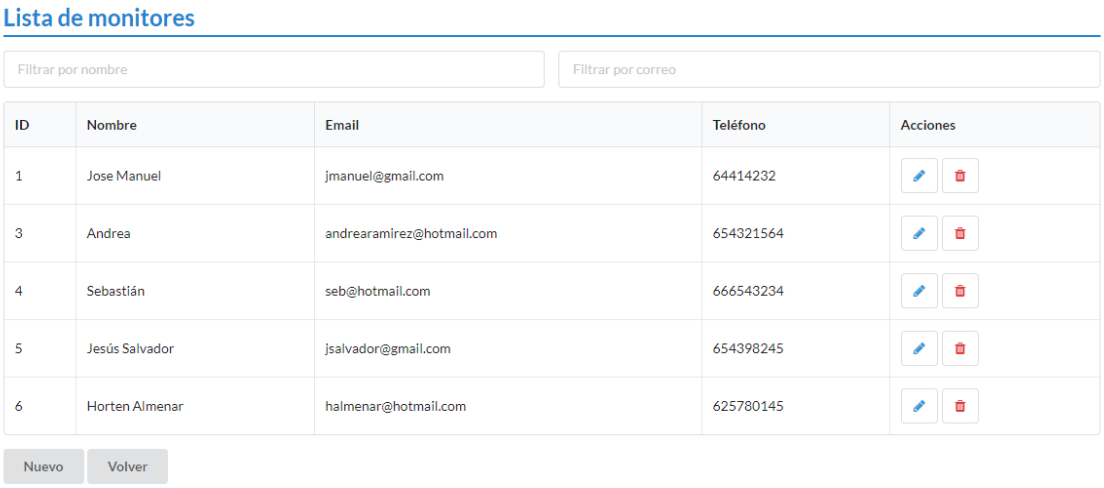


Ilustración 35: Vista - Listar monitores

En la sección de monitores, se dispone de un listado de los mismos (Ilustración 35), donde se va a permitir utilizar un filtro por nombre y por correo electrónico, crear un monitor nuevo, modificar monitores existentes, eliminar el monitor seleccionado o volver a la página anterior.

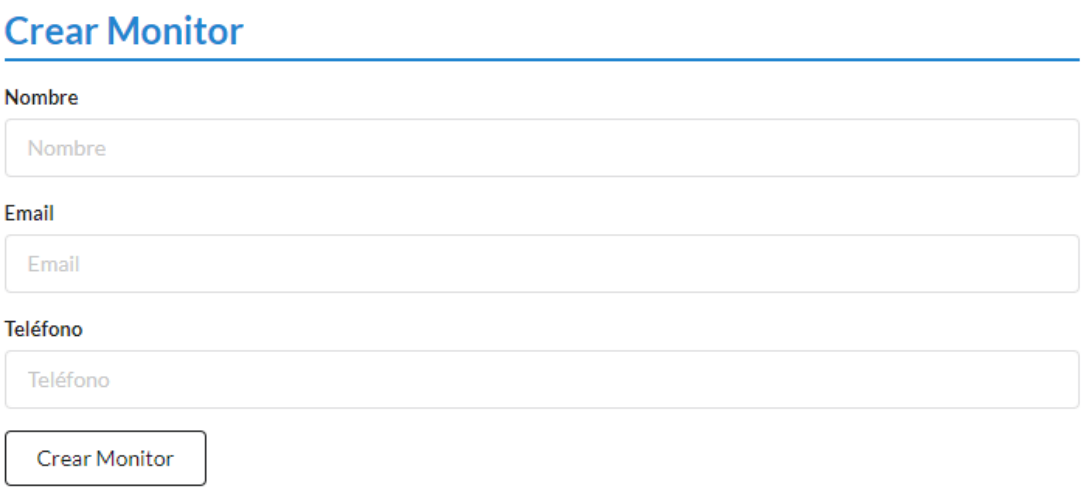


Ilustración 36: Vista - Crear monitor

Los datos a introducir en la creación de un monitor (Ilustración 36) son el Nombre, el correo electrónico y el teléfono.

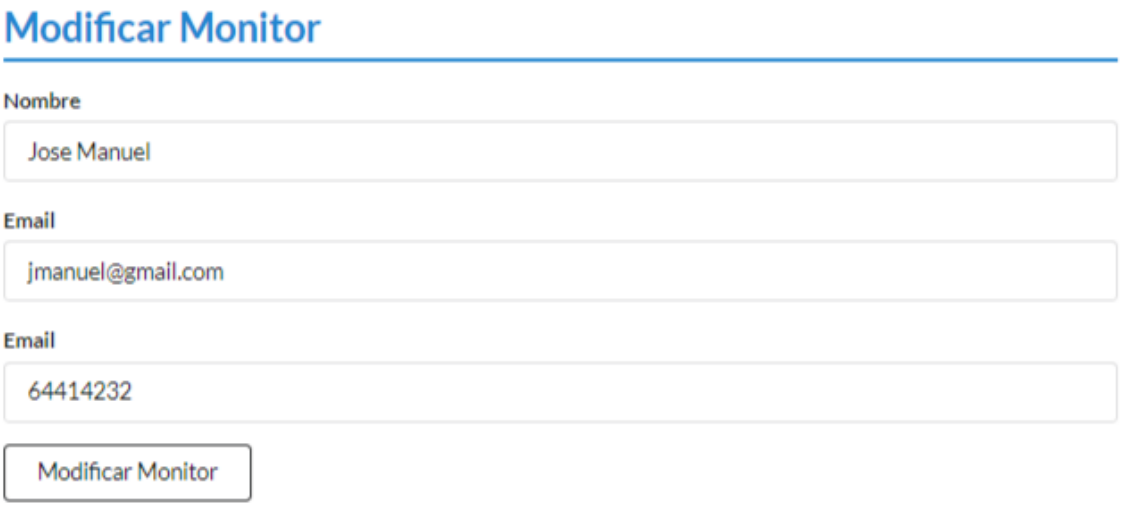


Ilustración 37: Vista - Modificar monitor

Al modificar un monitor (Ilustración 37) aparecerán los datos anteriores y se permitirá cambiar cualquiera por unos nuevos.

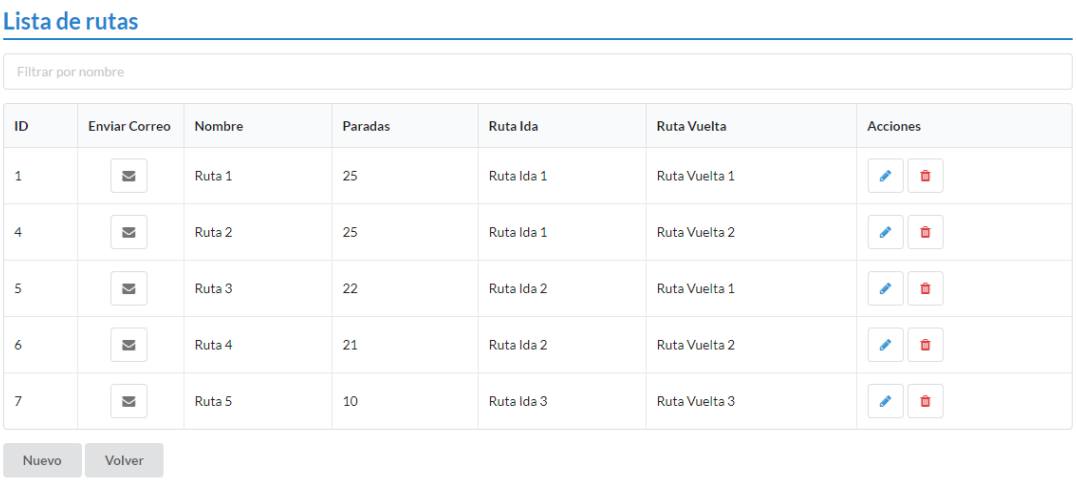


Ilustración 38: Vista - Listar rutas

La Ilustración 38 enseña la vista de listado de rutas disponibles. En esta tabla se muestra una acción nueva, la de enviar correo.

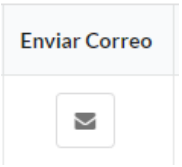


Ilustración 39: Vista - Enviar correo

Al utilizar el botón (Ilustración 39), se va a proceder a enviar un correo electrónico, avisando a los padres que tengan hijos en esa ruta de que se ha producido un atasco.

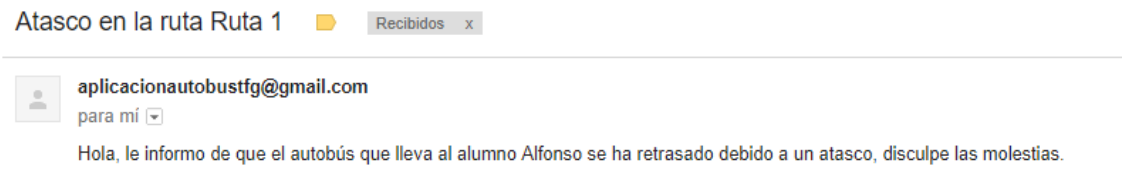


Ilustración 40: Vista - Correo enviado

En la Ilustración 40 se muestra el correo automático que se envía desde la cuenta aplicacionautobustfg@gmail.com al correo del padre de Alfonso informándole del atasco en la Ruta 1.

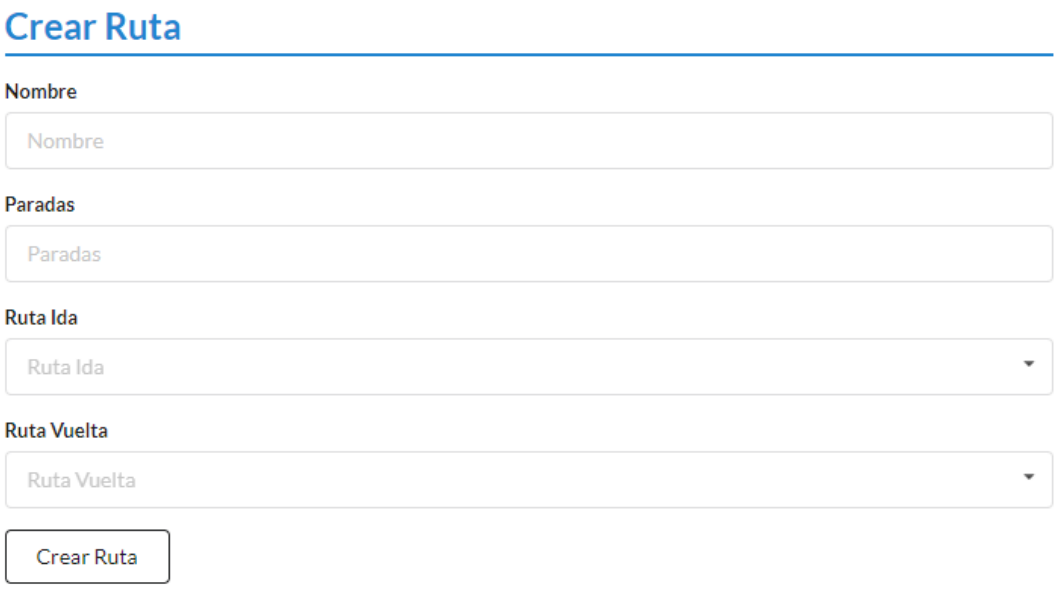


Ilustración 41: Vista - Crear ruta

Los datos a introducir en la creación de una ruta (Ilustración 41) son el nombre de la ruta, las paradas y la ruta de ida y de vuelta a seleccionar entre las creadas por el administrador.

Ilustración 42: Vista - Modificar ruta

Al modificar la ruta (Ilustración 42) se mantienen los datos introducidos con anterioridad, y se permite cambiar cualquiera de ellos.

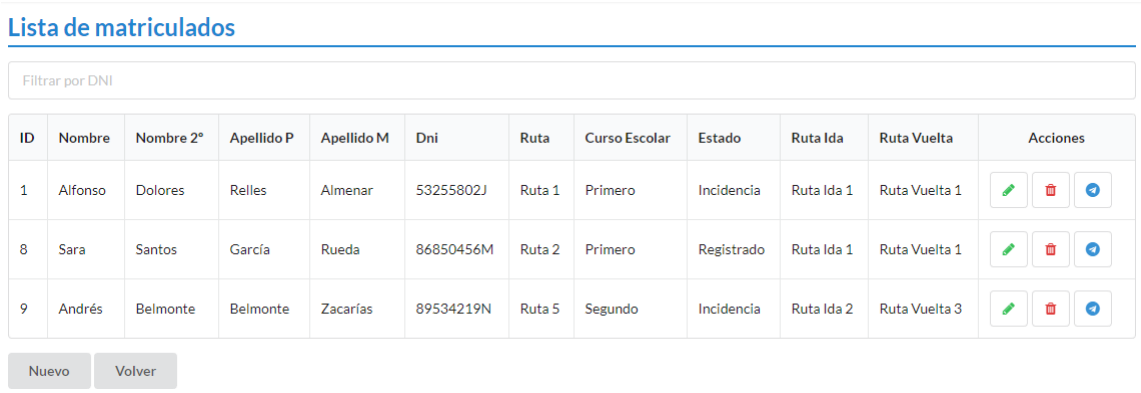


Ilustración 43: Vista - Listar matriculados

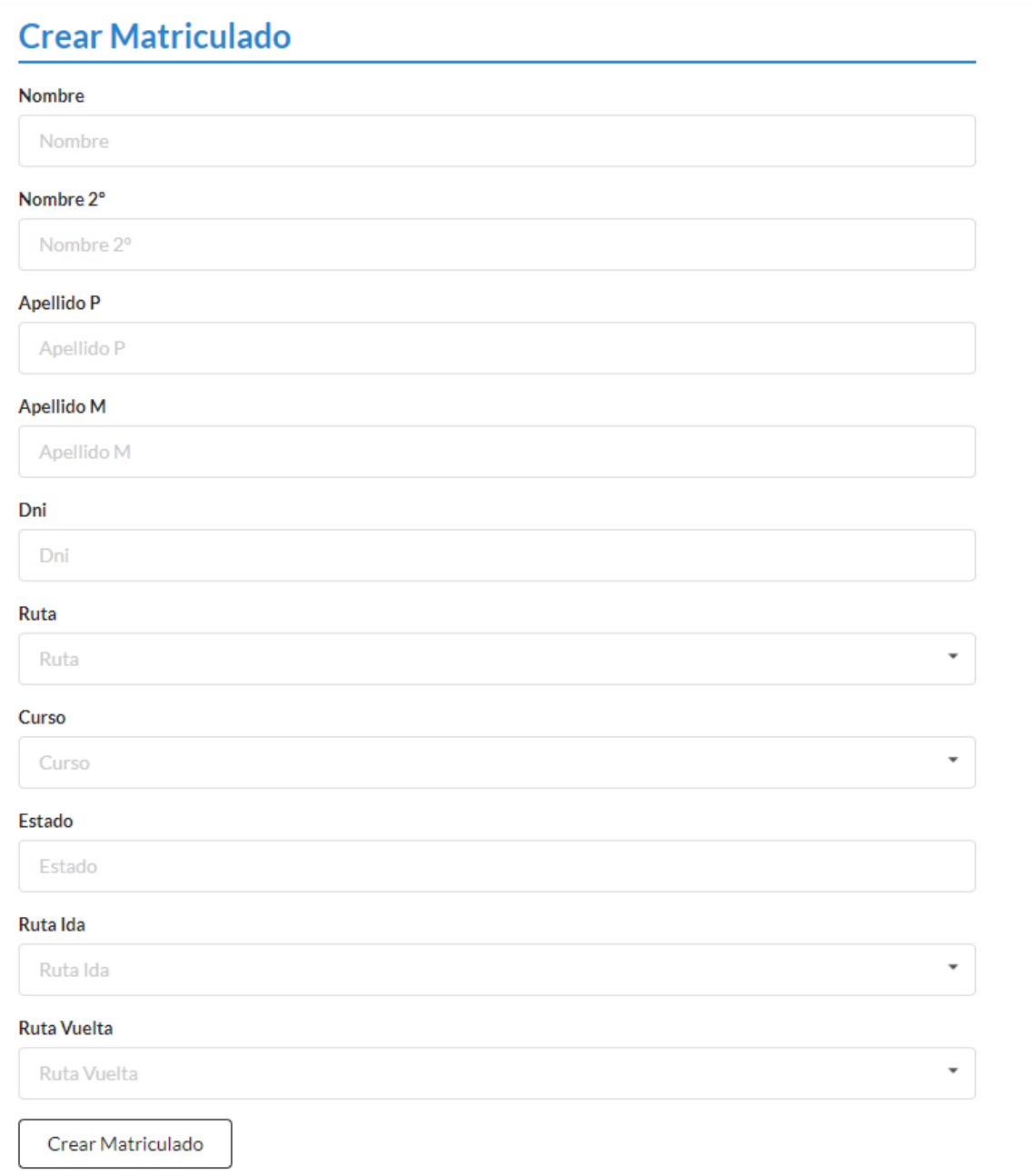


Ilustración 44: Vista - Crear matriculado

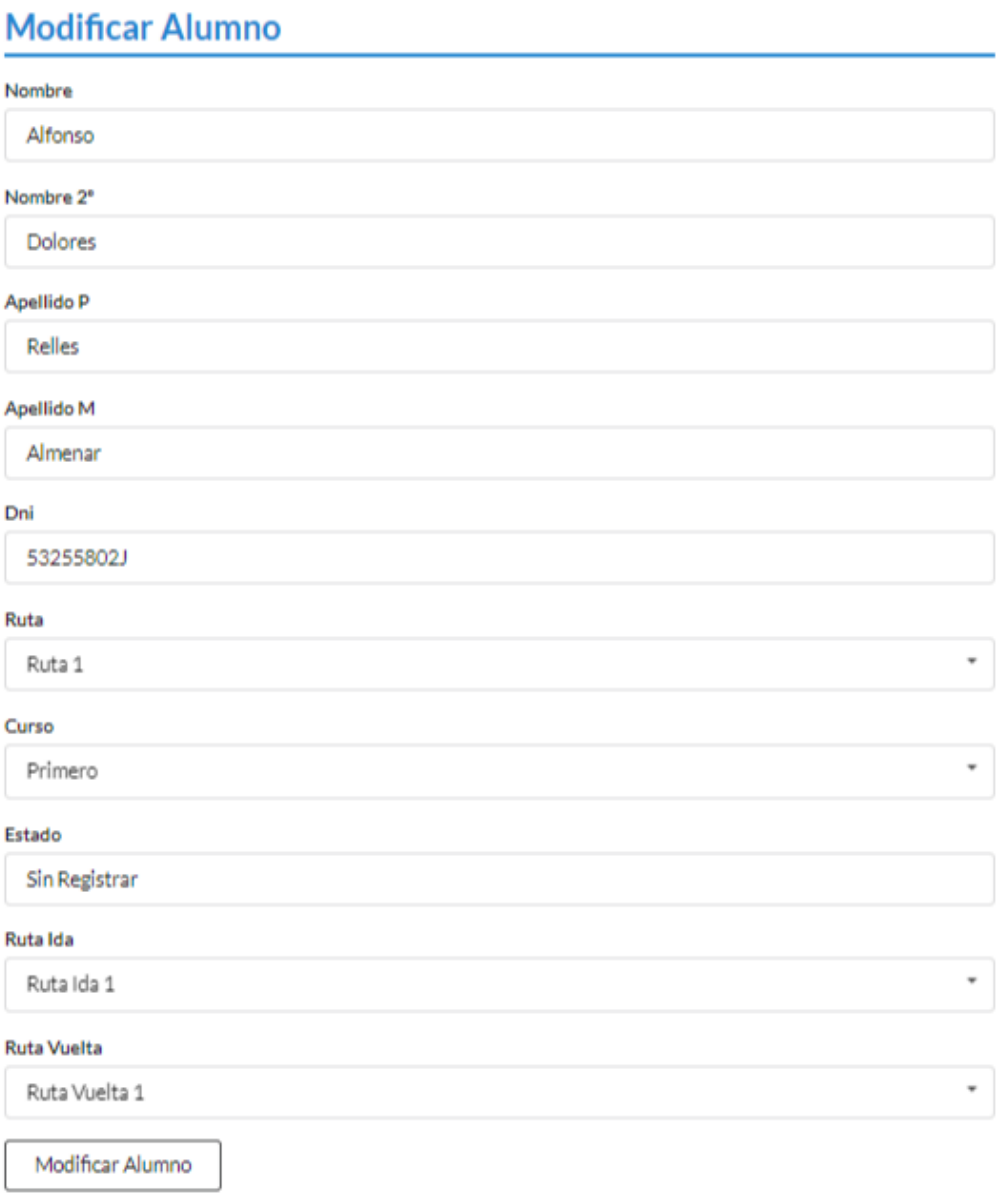


Ilustración 45: Vista - Modificar alumno

La vista de alumnos matriculados (Ilustración 43) muestra todos los alumnos registrados en la aplicación que están matriculados en el centro escolar. Las acciones disponibles en esta vista son las de filtrar por DNI del alumno, crear alumno nuevo, modificar alumnos existentes, eliminar alumnos y volver a la página anterior.

Al crear un alumno nuevo (Ilustración 44) se deberán introducir los datos siguientes:

- Nombre
- Segundo Nombre
- Apellido Paterno
- Apellido Materno
- DNI
- Ruta (campo seleccionable de rutas existentes)
- Curso (campo seleccionable de cursos existentes)
- Estado (campo seleccionable)
- Ruta de ida (campo seleccionable de rutas de ida existentes)
- Rutas de vuelta (campo seleccionable de rutas de vuelta existentes)

También se observa una nueva acción con el símbolo de la aplicación Telegram, la cual, al ser ejecutada, informará a los parientes del alumno mediante una notificación de Telegram el estado actual del alumno.

Se permite modificar alumnos ya creados (Ilustración 45). Aparecerán los datos que se introdujeron con anterioridad y se podrán cambiar por los valores deseados.

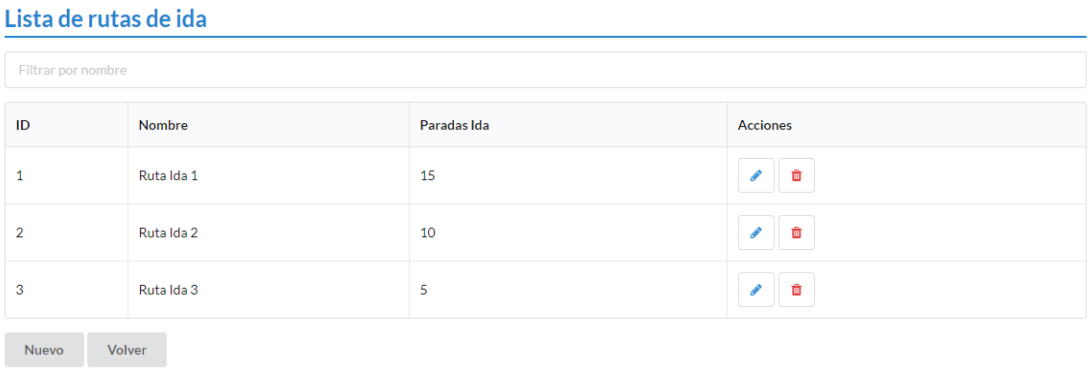


Ilustración 46: Vista - Listar rutas de ida

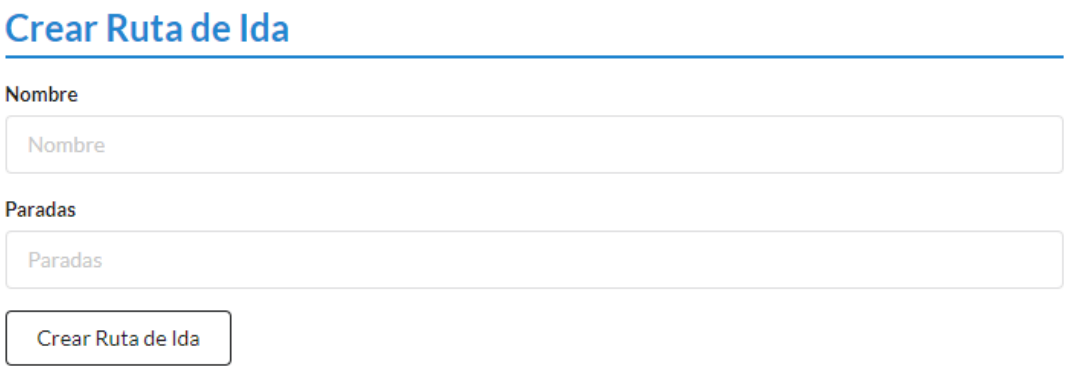


Ilustración 47: Vista - Crear rutas de ida

Ilustración 48: Vista - Modificar Ruta de ida

La ilustración 46 muestra la vista de rutas de ida. Las opciones disponibles van a ser las de filtrar por nombre de ruta de ida, creación de ruta de ida nueva, modificación de ruta de ida existente, eliminación de ruta existente y volver a la página anterior.

Al realizar la acción de “Nuevo” va a aparecer la vista de Crear ruta de ida (Ilustración 47) donde los datos a introducir son el nombre y el número de paradas.

La modificación de cualquier registro de rutas de ida (Ilustración 48) permite cambiar los valores asociados a una ruta existente.

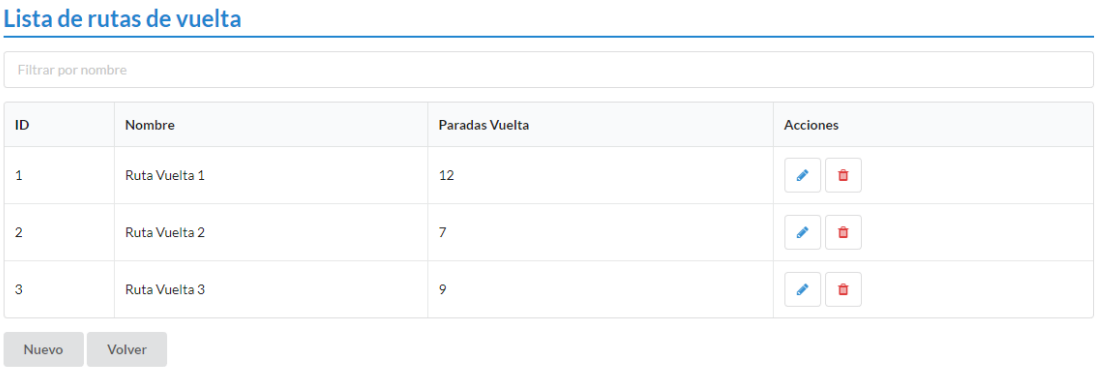


Ilustración 49: Vista - Listar rutas de vuelta

La ilustración 49 nos enseña la vista de rutas de vuelta. Las opciones disponibles son las de filtrar por nombre de ruta de vuelta, crear de ruta de vuelta nueva, modificar de ruta de vuelta existente, eliminación de ruta existente y volver a la página anterior.

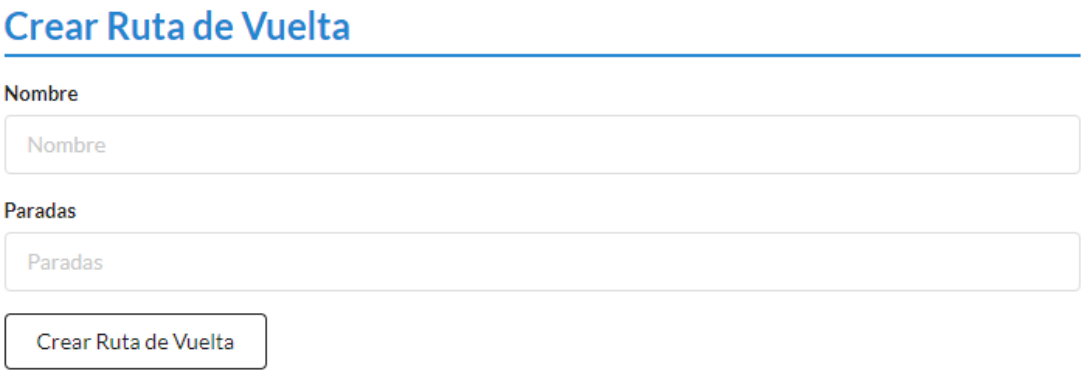


Ilustración 50: Vista - Crear ruta de vuelta

La ilustración 50 muestra la creación de rutas de vuelta, donde los datos a introducir son el Nombre de la ruta de vuelta y el número de paradas que tiene.

Ilustración 51: Vista - Modificar Ruta de vuelta

La modificación de registros de rutas de vuelta (Ilustración 51) proporciona el cambio de valores a registros de rutas de vuelta ya existentes en la base de datos.

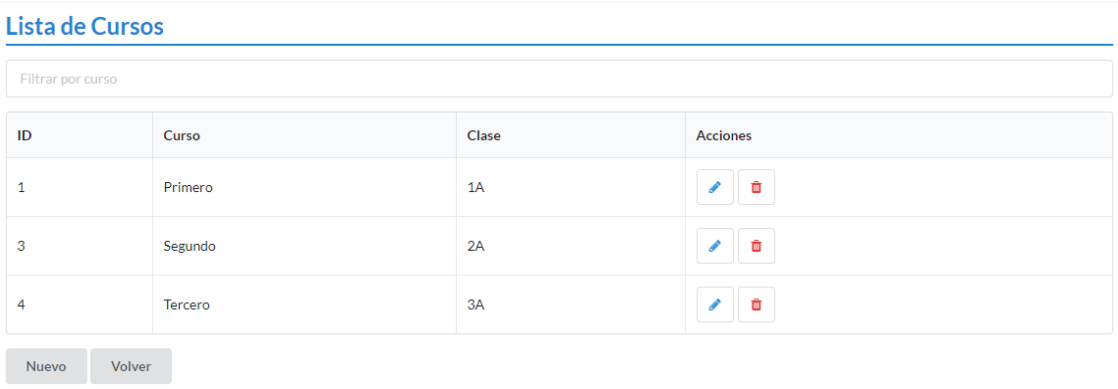


Ilustración 52: Vista - Listar cursos

La vista de cursos escolares (Ilustración 52) muestra un listado de cursos creados por el administrador desde la aplicación. Las opciones disponibles en esta vista son las de filtrar por curso, crear curso escolar nuevo, modificar curso existente, eliminar curso y volver a la página anterior.

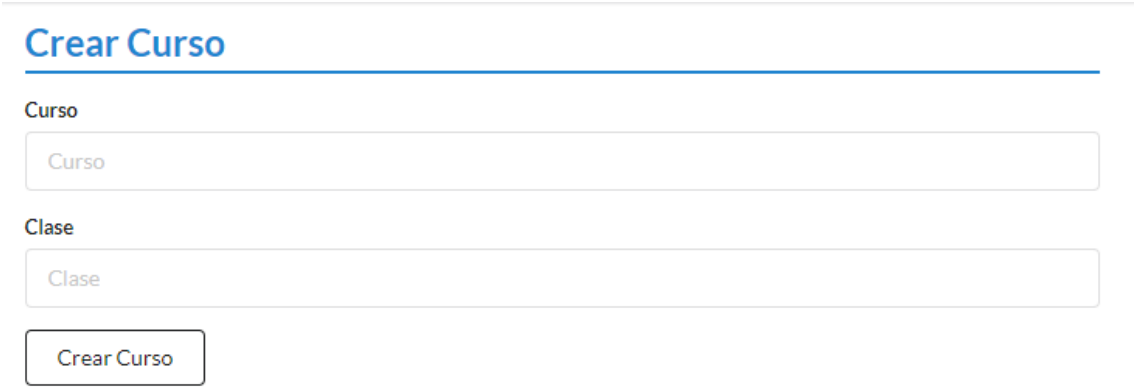


Ilustración 53: Vista - Crear curso

La vista de creación de curso (Ilustración 53) permite crear un nuevo registro de cursos escolares, introduciendo como datos el curso y la clase.

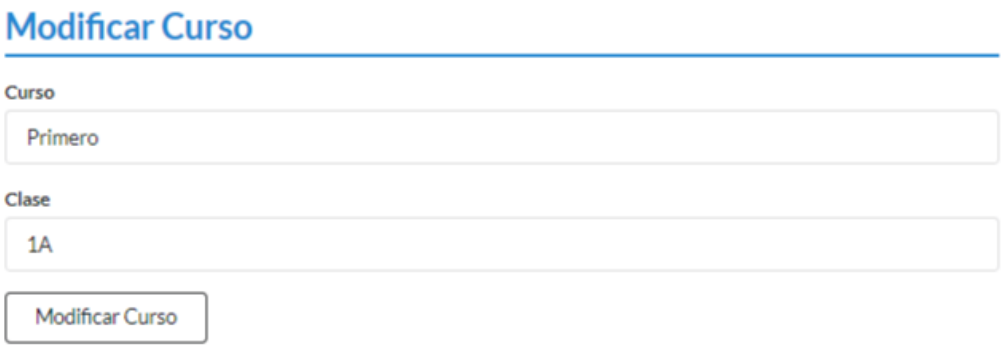


Ilustración 54: Vista - Modificar curso

La Ilustración 54 enseña la vista de modificación de curso existente, en la cual se puede cambiar cualquier dato anterior por nuevos.

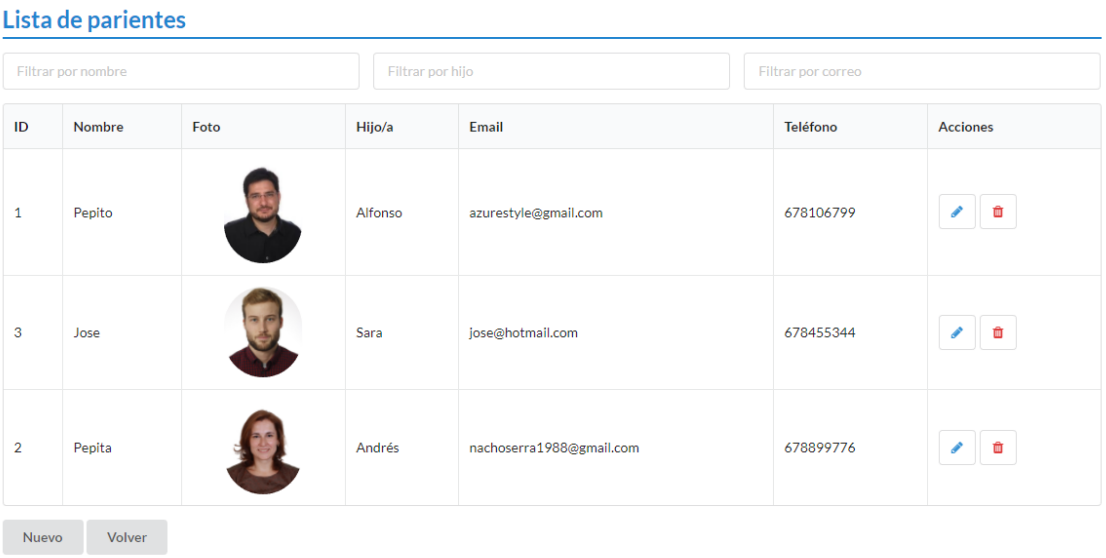


Ilustración 55: Vista - Listar parientes

En la Ilustración 55 tenemos el listado de parientes, donde se muestran a todos los padres registrados en la aplicación por el administrador, junto con su foto, guardada en un fichero estático en el servidor donde se encuentra instalada la aplicación. Las opciones disponibles en esta vista son las de filtrar por nombre del padre, hijo, o correo electrónico, creación de nuevo pariente, modificación de pariente existente, eliminación de pariente y volver a la página anterior.

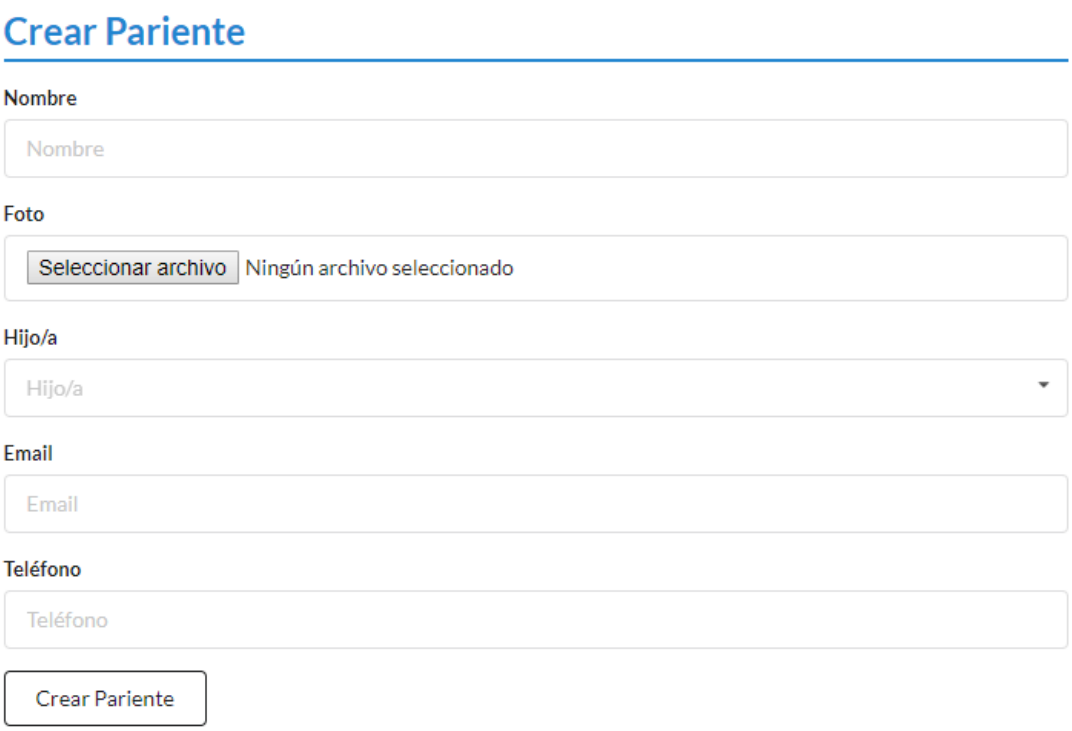


Ilustración 56: Vista - Crear pariente

En la creación de un pariente nuevo (Ilustración 56) los datos a introducir son los siguientes:

- Nombre del pariente
- Foto del pariente
- Hijo del pariente (Campo seleccionable)
- Correo electrónico
- Teléfono

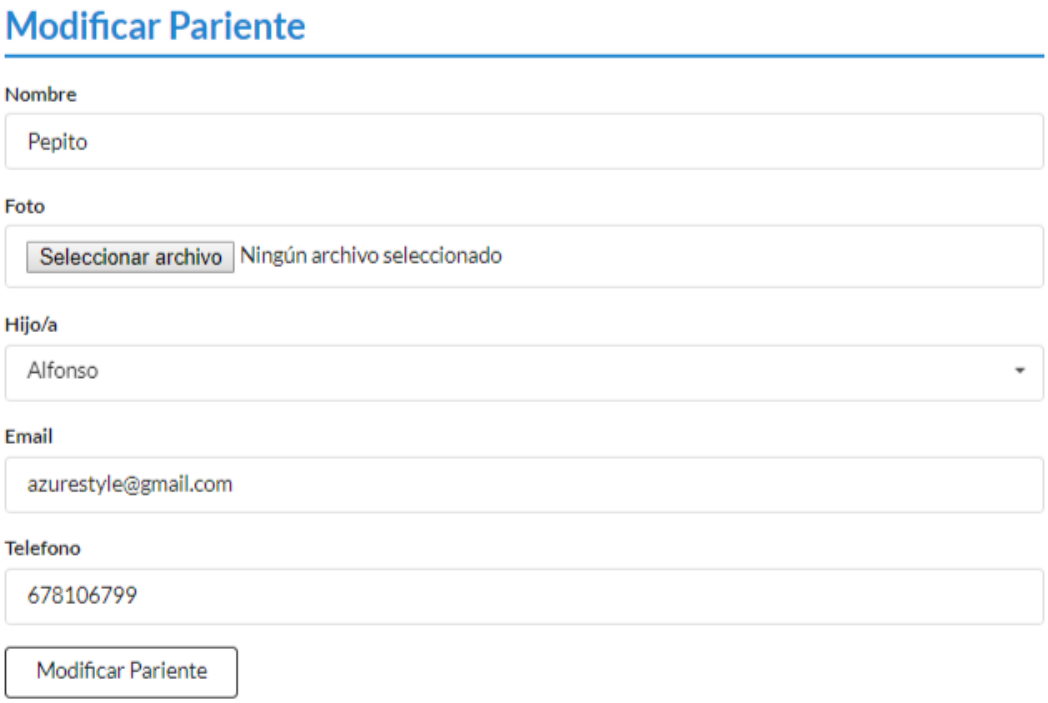


Ilustración 57: Vista - Modificar pariente

La modificación del pariente (Ilustración 57) permite cambiar valores anteriores por datos nuevos.

Ilustración 58: Vista - Listar paradas

En el apartado de paradas, se muestra el listado de todas las paradas disponibles (Ilustración 58). Las opciones en esta vista son las de filtrar por nombre y por ruta, crear parada nueva, modificar parada existente, eliminar parada existente y volver a la página anterior.

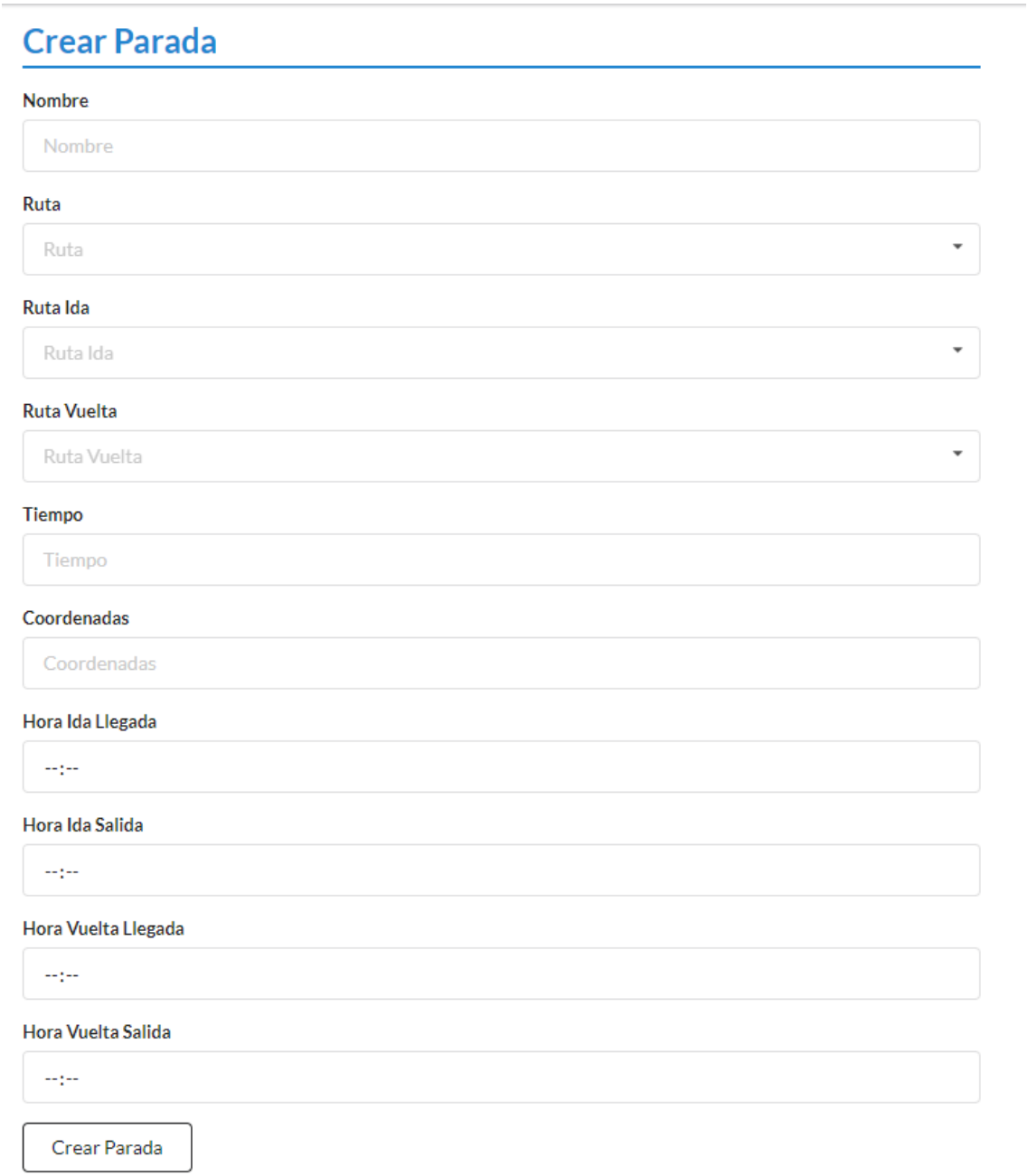


Ilustración 59: Vista - Crear parada

En la creación de una parada nueva (Ilustración 59) los datos a introducir son el nombre de la parada, la ruta, ruta de ida y rutas de vuelta seleccionables, el tiempo de parada del autobús, las coordenadas de la parada, la hora de ida de llegada, la hora de ida de salida, la hora de vuelta de llegada y la hora de vuelta de salida.

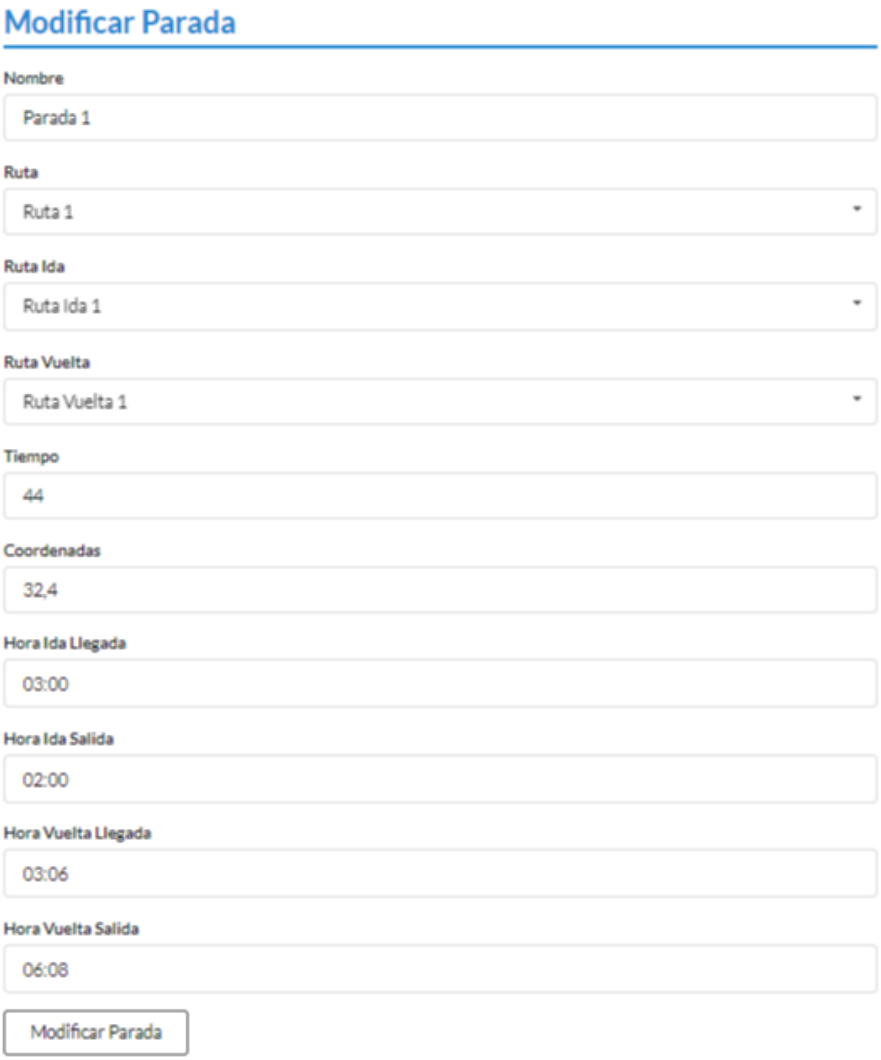


Ilustración 60: Vista - Modificar parada

La vista de modificación de parada (Ilustración 60) da la posibilidad de modificar todos los datos existentes en la base de datos por unos nuevos.

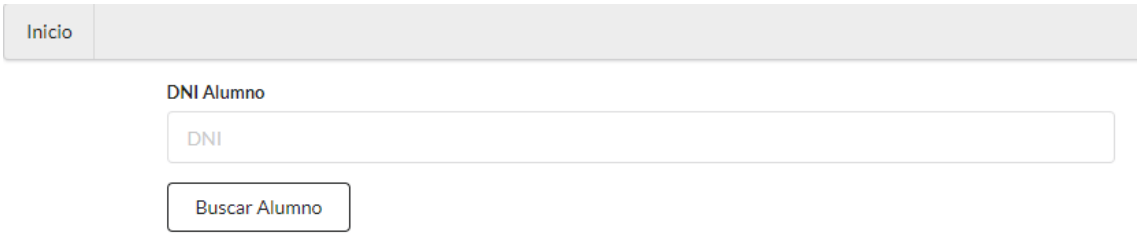


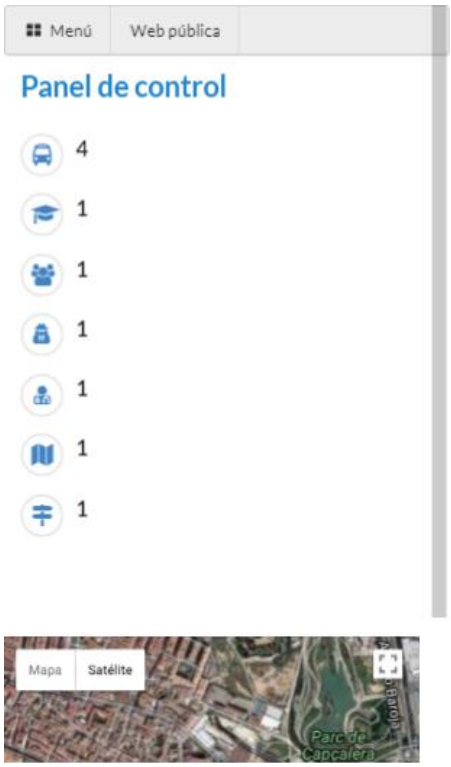
Ilustración 61: Vista - Página pública - Buscar Alumno

En cuanto al apartado público (Ilustración 61) tenemos un formulario en el que los parientes pueden buscar a su hijo introduciendo el DNI del alumno.

Nombre	Nombre 2º	Apellido P	Apellido M	Dni	Estado
Alfonso	Dolores	Relles	Almenar	53255802J	Sin Registrar

Ilustración 62: Vista - Página pública - Estado del alumno

Una vez introducidos los datos, aparecerá una tabla con los datos del alumno y su estado, así como un mapa de *Google Maps* indicando su posición actual.



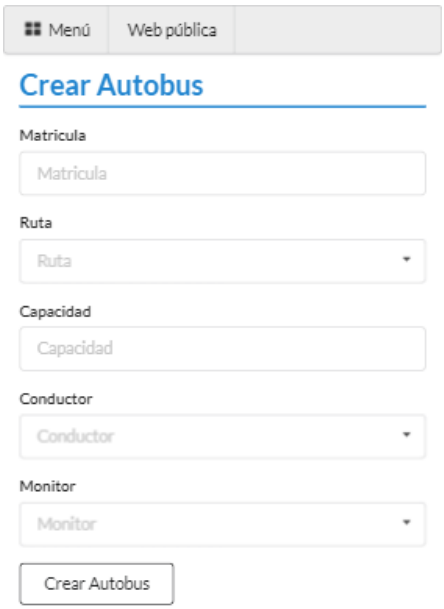
En cuanto al apartado móvil de la aplicación, gracias al *framework* Semantic UI, la aplicación ofrece un diseño móvil *responsive*, es decir, se adapta a la pantalla del dispositivo y permite realizar las mismas opciones que desde un navegador web.

En la Ilustración 63 tenemos la página de inicio de la vista móvil, donde tenemos toda la información sin tener que deslizar la imagen lateralmente.

Ilustración 63: Vista móvil - Inicio

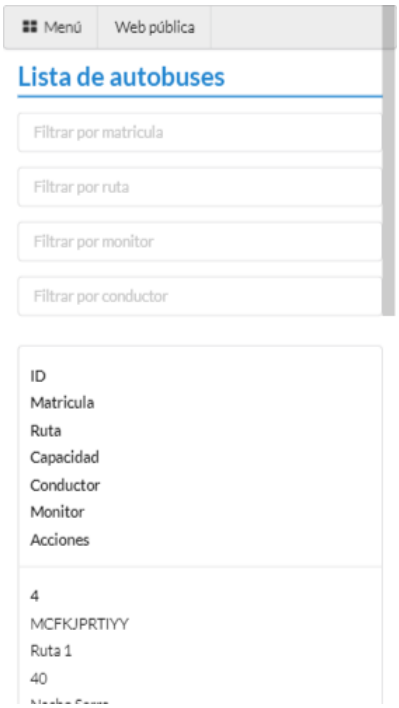
La Ilustración 64 enseña el menú lateral desplegable que aparece en la vista móvil, ofreciendo la misma funcionalidad que desde el navegador web pero con la vista adaptada al nuevo dispositivo.

Ilustración 64 - Vista móvil - Menú desplegable



La creación del autobús desde el móvil (Ilustración 65), dispone de toda la información en el centro sin necesidad de moverse lateralmente por la pantalla del dispositivo.

Ilustración 65: Vista móvil - Crear autobús



La Ilustración 66 enseña la vista móvil del listado de autobuses y su diseño *responsive*. Adaptado a la pantalla del dispositivo muestra las tablas y los atributos del objeto de manera que se puedan leer de arriba hacia abajo.

Ilustración 66: Vista móvil - Listado de autobuses

5. Implementación

En este apartado se van a explicar las tecnologías utilizadas en el desarrollo de la aplicación y el código utilizado para el correcto funcionamiento de la misma.

5.1 Tecnologías utilizadas

5.1.1 Express.js

Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.

Su uso ofrece métodos tales como manejador de sesiones y *routing*, es decir, es quien se encarga de la manera en la que una aplicación responde a una solicitud del cliente.

5.1.2 Node.js

Node.js [9] es una plataforma de código abierto para la capa del servidor basada en el motor de JavaScript V8 de Google.

Está catalogado como un marco orientado a eventos asíncronos y su principal objetivo es construir aplicaciones de red altamente escalables (p.ej. servidores web). Node.js basa su funcionalidad en módulos, los cuales pueden ser compilados en el propio binario o pueden ser módulos de terceros.

Los motivos principales de su uso en este proyecto han sido por su capacidad de ofrecer eventos asíncronos y entrada y salida de datos no bloqueantes, por lo que ha permitido realizar un servidor sencillo que escuche en una dirección IP y un puerto dado. Otro motivo ha sido el de aprender y conocer esta tecnología que cada vez se está utilizando más.

5.1.3 Jade

Jade es un motor de plantillas [13] que simplifica la sintaxis de html, agiliza y facilita el proceso de desarrollo de plantillas html.

Para poder trabajar con Jade es necesario tener instalado Node.js.

Los motivos de su uso han sido los siguientes:

- Es el motor de plantillas principal de nuestra primera tecnología (Node.js)
- Simplifica las etiquetas html, de tal forma que no es necesario utilizarlas al principio y al final, ahorrando mucho tiempo en la escritura de código.
- La anidación de elementos se basa en la identación, ahorrando tener que cerrar etiquetas, lo que provocaba en muchos casos errores de maquetación y problemas con CSS.

5.1.4 CSS

CSS o hojas de estilo en cascada (*Cascading Style Sheets*) [11] es el lenguaje utilizado para describir la presentación de documentos HTML o XML].

CSS describe como debe ser renderizado el elemento estructurado en pantalla, es decir, de su estilo.

Se ha usado CSS por su capacidad de separar el contenido de la presentación, proporcionando flexibilidad y optimización de los tiempos de carga.

5.1.5 MySQL

MySQL es un sistema de gestión de base de datos relacional [10]. Está considerada como la base de datos open source más popular del mundo, sobre todo para entornos de desarrollo web.

Los motivos de su uso han sido la escalabilidad y su facilidad de aprender, ya que con conocer el estándar SQL se puede manejar una base de datos MySQL sin ningún problema.

5.1.6 JQuery

JQuery [17] es una biblioteca multiplataforma de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX, la cual también hacemos uso en nuestra aplicación.

El motivo principal para utilizar JQuery es la rapidez y comodidad al escribir código. También, al ser de uso generalizado encontramos mucha comunidad, *plugins* y documentación.

5.1.7 Semantic UI

Semantic UI [12] es un *framework* que se usa para crear el diseño de interfaces de manera *responsive* utilizando HTML y CSS.

Una de las grandes diferencias con el resto de *frameworks* es la utilización del lenguaje natural, es decir, el uso de sintaxis para crear componentes de manera legible. Esto hace que el desarrollo sea más intuitivo y rápido.

5.2 Aplicación Web

5.2.1 Configuración

Para poder utilizar nuestra aplicación web es necesario configurar correctamente el acceso a la base de datos.

```
var config = {
  host : 'localhost',
  user : 'root',
  password : '',
  database : 'autobusapp',
  dateStrings : true


}

module.exports = config;
```

Ilustración 67: Implementación - Configuración

Node.js nos permite configurar el acceso a la base de datos mediante un archivo llamado config.js [3].

En él tenemos los siguientes atributos:

- **Host**: dirección donde se va a ejecutar la aplicación web, en nuestro caso es localhost, ya que se está ejecutando en un entorno local.
- **User:** usuario con el que se accede a nuestra base de datos
- **Password:** contraseña usada junto al usuario para poder acceder a la base de datos. En nuestro caso está vacía porque no dispone de contraseña.
- **Database:** nombre que se le ha dado a nuestra base de datos.
- **DateStrings:** si es *true* se avisa de que existen cadenas de texto de formato fecha en la base de datos.

Si los datos introducidos en el fichero son correctos, la aplicación se conectará a nuestra base de datos.

5.2.2 Rutas

Las rutas son las que determinan como responde la aplicación a una solicitud de cliente. Es decir, cada vez que el usuario "navega" por nuestra aplicación por las diferentes vistas, Express.js, mediante su sistema de enrutamiento ofrece diferentes funciones dependiendo de donde se encuentre el usuario en la aplicación.

```
//rutas para autobus
router.get('/autobus', controllers.autobuscontroller.getAutobus);
router.get('/nuevo', controllers.autobuscontroller.getNuevoAutobus);
router.post('/crearautobus', controllers.autobuscontroller.postNuevoAutobus);
router.post('/eliminarautobus', controllers.autobuscontroller.eliminarAutobus);
router.get('/modificar/:id', controllers.autobuscontroller.getModificarAutobus);
router.post('/editar', controllers.autobuscontroller.postModificarAutobus);
```

Ilustración 68: Implementación - Rutas Autobús

En la Ilustración 68 tenemos de ejemplo las rutas utilizadas para el objeto autobús:

- /autobus: cuando la aplicación tenga esta ruta en la URL, se llamará a la función getAutobus del controlador autobuscontroller.
- /nuevo: se llamará a la función getNuevoAutobus.
- /crearautobus: llama a la función postNuevoAutobus.
- /eliminarautobus: llama a la función eliminarAutobus
- /modificar/:id: llama a la función getModificarAutobus utilizando como parámetro la id del autobús.
- /editar: llama a la función postModificarAutobus.

En la sección de controladores se hablará de las funciones anteriormente citadas.

5.2.3 Controladores

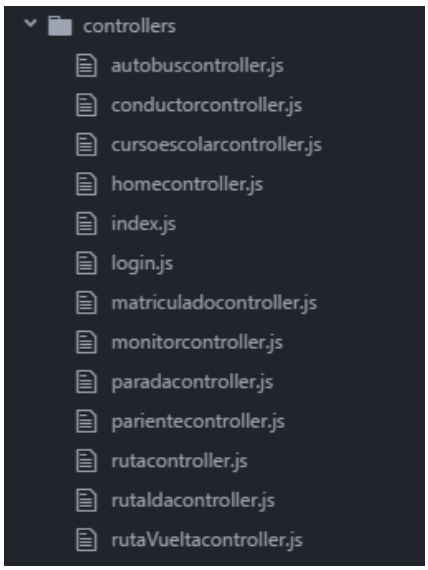


Ilustración 69: Implementación - Controladores

Todos los controladores [4] se encuentran en la carpeta *controllers* como se muestra en la Ilustración 69. Van a contener las funciones a las que hacen referencia las rutas. Para explicar su funcionamiento se va a seguir utilizando de ejemplo el controlador del autobús, en concreto la función "getAutobus".

```
//funciones del controlador
getAutobus : function(req, res, next){
  var config = require('.././database/config');
  var db = mysql.createConnection(config);
  db.connect();

  var autobus = null;
  db.query('SELECT * FROM autobus a,conductor c, monitor m, \
            ruta r where a.idconductor = c.idconductor and \
            a.idmonitor = m.idmonitor and a.idruta = r.idruta',
            function(err, rows, fields){

      if(err) throw err;
      autobus = rows;

      db.end();
      //renderizamos la vista autobus.jade
      // y le pasamos el atributo autobus que son las rows
      res.render('autobus/autobus', {autobus : autobus});
  });
},
```

Ilustración 70: Implementación – controlador autobus - getAutobus

La Ilustración 70 muestra la función “getAutobus”, la cual se utiliza cuando la ruta del navegador es /autobus. Lo primero que se realiza es la carga de la configuración de la base de datos en la variable “config”, seguida de la conexión con la base de datos mediante la función createConnection(config) y db.connect(). A partir de aquí podemos realizar cualquier tipo de consulta SQL.

En nuestro caso gracias a la función db.query() la consulta que se realiza va a guardar en la variable “autobus” todos los autobuses que existen actualmente, así como sus atributos, los atributos del monitor asociado, los atributos del conductor asociado y los atributos de la ruta asociada al autobús.

Por último, se va a finalizar la conexión de la base de datos y se va a utilizar la función res.render(‘autobus/autobus’, {autobus : autobus}) , la cual va a mostrar nuestra vista creada “autobus”, pasándole por contexto la variable “autobus”, que contiene todos los datos que se quieren mostrar en la vista.

5.2.4 Vistas

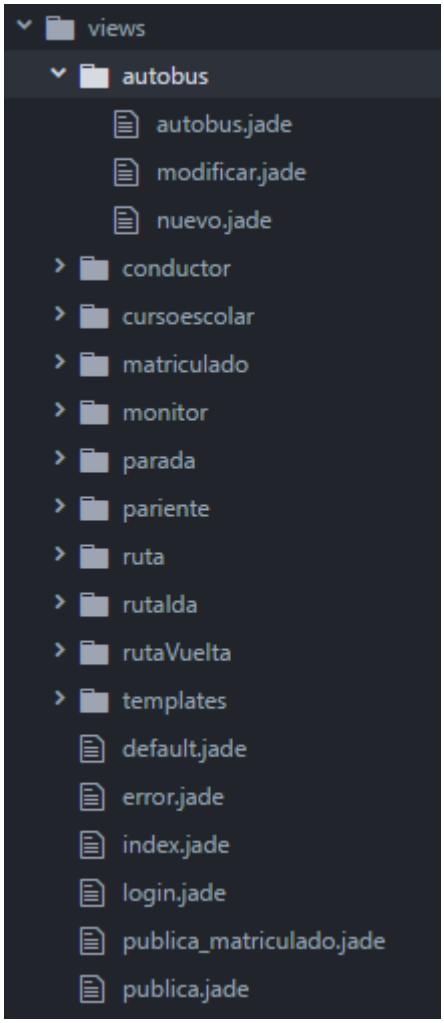


Ilustración 71: Implementación - Vistas

Las vistas se encuentran ordenadas por cada elemento en la carpeta *views* (Ilustración 71)

Siguiendo el ejemplo, para explicar este apartado se va a hacer uso de la vista “autobus”.

```
h2(class=['ui','header','blue','dividing'])= 'Lista de autobuses'
div(class=['ui','form'])
  div(class=['ui','four','fields'])
    div(class=['ui','field'])
      input(type='text' name="filtro" onkeyup="myFunction()" placeholder='Filtrar por matricula' id="filtro")
    div(class=['ui','field'])
      input(type='text' name="filtro" onkeyup="myFunction2()" placeholder='Filtrar por ruta' id="filtroRuta")

    div(class=['ui','field'])
      input(type='text' name="filtro" onkeyup="myFunction3()" placeholder='Filtrar por monitor' id="filtroConductor")

    div(class=['ui','field'])
      input(type='text' name="filtro" onkeyup="myFunction4()" placeholder='Filtrar por conductor' id="filtroMonitor")

table(class=['ui','table','celled'] id='tbl-autobus')
  thead
    tr
      th= 'ID'
      th= 'Matricula'
      th= 'Ruta'
      th= 'Capacidad'
      th= 'Conductor'
      th= 'Monitor'
      th= 'Acciones'

  tbody
    each bus in autobus
      tr
        td(id='id')= bus.id
        td= bus.matricula
        td= bus.nombreRuta
        td= bus.capacidad
        td= bus.nombreConductor
        td= bus.nombreMonitor
        td
          a(class=['ui button basic icon '] href='/modificar/'+bus.id data-inverted='' data-tooltip='Modificar' data-position='top center')
            i(class=['write icon blue'])

          a(class=['ui button basic icon '] href='#' data-inverted='' data-tooltip='Eliminar' data-position='top center' id='btn-eliminar')
            i(class=['trash icon red'])
a(class=['ui', 'button' ] href='/nuevo')= 'Nuevo'
```

Ilustración 72: Implementación - Vista autobús

En la primera parte de la Ilustración 72 se pueden observar los cuatro filtros que tiene la vista “autobus”, el filtro por matricula, por ruta, por monitor y por conductor.

La parte que hay que destacar es la creación de la tabla con los atributos del autobús. Como se contó en el anterior apartado, a esta vista se le pasó por contexto la variable “autobus”, y como vemos, hacemos uso de ella al rellenar los datos de la tabla.

“Each bus in autobus” va a recorrer todos los autobuses existentes en el momento, asignando cada uno a la variable “bus”, por lo que usando “bus.matricula”, vamos a acceder al atributo matricula de ese autobús en concreto.

5.2.5 Añadir elemento

En este apartado se va a explicar la función de añadir elemento, utilizando como ejemplo el elemento autobús.

```
getNuevoAutobus : function(req, res, next){

  var config = require('.././database/config');
  var db = mysql.createConnection(config);
  db.connect();

  db.query('SELECT * from conductor', function(err, rows, fields){
      if(err) throw err;
      var conductor = rows;

      db.query('SELECT * from monitor;', function(err, rows, fields){
          if(err) throw err;
          var monitor = rows;

          db.query('SELECT * from ruta;', function(err, rows, fields){
              if(err) throw err;
              var ruta = rows;

              db.end();
              res.render('autobus/nuevo', {monitor : monitor, conductor : conductor, ruta : ruta});
          });

      });
  });

},
```

Ilustración 73: Implementación - Añadir autobús - Método GET

En primer lugar (Ilustración 73), como un autobús tiene un conductor, un monitor, y una ruta relacionada, los cuales son también elementos principales, es necesario hacer tres consultas SQL.

El resultado de estas consultas se va a guardar en las variables “conductor”, “monitor” y “ruta” respectivamente. Por último, se va a preparar la vista “/nuevo” pasando por contexto los datos obtenidos. De este modo se visualizará la vista de la Ilustración 29, donde podemos seleccionar los datos que va a utilizar el autobús y cuyo código es el de la Ilustración 74.

```
block content
  include ../templates/nav
  div(class=['ui','container','grid','stackable'])
    div(class=['ten','wide','column'])
      h2(class=['ui','header','blue','dividing'])= 'Crear Autobus'

      div(class='form-nuevoautobus')
        form(class=['ui','form'] action='/crearautobus' method='post')
          div(class=['ui','field'])
            label(for='matricula')= 'Matricula'
            input(type='text' name="matricula" placeholder='Matricula')

          div(class=['ui','field'])
            label(for='idruta')= 'Ruta'
            select(class=['ui', 'dropdown'] name="idruta")
              option(value="" disabled selected hidden)= 'Ruta'
              each ru in ruta
                option(value=ru.idruta)= ru.nombreRuta
            //- input(type='text' name="idruta" placeholder='Ruta')
          div(class=['ui','field'])
            label(for='capacidad')= 'Capacidad'
            input(type='text' name="capacidad" placeholder='Capacidad')
          div(class=['ui','field'])
            label(for='idconductor')= 'Conductor'
            select(class=['ui', 'dropdown'] name="idconductor")
              option(value="" disabled selected hidden)= 'Conductor'
              each con in conductor
                option(value=con.idconductor)= con.nombreConductor
            //- input(type='text' name="idconductor" placeholder='Conductor')
          div(class=['ui','field'])
            label(for='idmonitor')= 'Monitor'
            select(class=['ui', 'dropdown'] name="idmonitor")
              option(value="" disabled selected hidden)= 'Monitor'
              each mon in monitor
                option(value=mon.idmonitor)= mon.nombreMonitor
            //- input(type='text' name="idmonitor" placeholder='Monitor')
          button(class=['ui','button','basic','black'])= 'Crear Autobus'
          div(class=['ui','message','error'])
```

Ilustración 74: Implementación - Nuevo autobús - Entrada de datos

A destacar en el código de la entrada de datos del nuevo autobús, la acción que se realizará al finalizar el formulario es "/crearautobus" y el método utilizado será POST. Lo que quiere decir que una vez introducidos los datos y se pulse el botón de crear autobús, se va a llamar a la función "postNuevoAutobus", cuyo código es el siguiente (Ilustración 75).

```
postNuevoAutobus : function(req, res, next){
  var fechaactual = new Date();
  var fecha = dateFormat(fechaactual, 'yyyy-mm-dd h:MM:ss');

  var autobus = {
    matricula : req.body.matricula,
    idruta : req.body.idruta,
    capacidad : req.body.capacidad,
    idconductor : req.body.idconductor,
    idmonitor : req.body.idmonitor
  };
  var config = require('.././database/config');
  var db = mysql.createConnection(config);
  db.connect();
  db.query('INSERT INTO autobus SET ?', autobus, function(err,rows,fields){
    if(err) throw err;
    db.end();
  });
  res.render('autobus/nuevo', {info : 'Autobus creado correctamente', ruta : autobus.idruta,
  monitor : autobus.idmonitor, conductor : autobus.idconductor});
```

Ilustración 75: Implementación - Crear autobús - Método POST

De la Ilustración 75 se destaca la variable “autobus” donde se está guardando un diccionario con clave y valor. En este caso la clave es el mismo nombre de atributo que contiene la base de datos, y el valor, el que se introdujo en la vista de creación del autobús, el cual se recoge mediante el objeto “req.body”, que contiene todos los parámetros de la vista enviados por el cliente como parte de la solicitud POST.

A continuación, se crea una conexión con la base de datos como se explicó anteriormente, y se inserta un registro nuevo en la base de datos, pasando como datos los que se recogieron en la variable “autobus”.

5.2.6 Modificar elemento

En esta sección, se va a explicar la parte del código que permite realizar modificaciones de un elemento de la base de datos, en este caso se va a continuar con el ejemplo del autobús.

```
getModificarAutobus : function(req, res, next){
  var id = req.params.id;
  var config = require('.././database/config');

  var db = mysql.createConnection(config);

  db.connect();

  var autobus = null;

  db.query('SELECT * FROM autobus a, conductor c, monitor m, \
   ruta r where a.id = ? and a.idruta=r.idruta and a.idmonitor = m.idmonitor \
   and a.idconductor = c.idconductor ',id,function(err,rows,fields){
    if(err) throw err;

    var autobus = rows;

    db.query('SELECT * from conductor', function(err, rows, fields){
        if(err) throw err;
        var conductor = rows;


        db.query('SELECT * from monitor;', function(err, rows, fields){
            if(err) throw err;
            var monitor = rows;

            db.query('SELECT * from ruta;', function(err, rows, fields){
                if(err) throw err;
                var ruta = rows;


                db.end();
                res.render('autobus/modificar', {autobus: autobus, ruta:ruta, monitor:monitor, conductor:conductor});
              });

         });
     });

  });
},
```

Ilustración 76: Implementación - Modificar autobús – Método GET

A diferencia del método de crear elemento, lo primero que se trata de conseguir es la "id" del autobús seleccionado (Ilustración 76). Al ser un atributo único nos aseguramos de que se va a modificar el autobús que se desea. A continuación, se realizan las consultas necesarias y se pasan por contexto las variables que contendrá nuestra vista ("autobus", "ruta", "monitor" y "conductor").

```
extends ../default

block content
  include ../templates/nav
  div(class=['ui','container','grid','stackable'])
    div(class=['ten','wide','column'])
      h2(class=['ui','header','blue','dividing'])= 'Modificar Autobús'

      div(class='form-nuevoautobus')
        form(class=['ui','form'] action='/editar' method='post')
          div(class=['ui','field'])
            label(for='matricula')= 'Matricula'
            input(type='text' name="matricula" placeholder='Matricula' value=autobus[0].matricula)
          div(class=['ui','two','fields'])
            div(class=['ui','field'])
              label(for='idruta')= 'Ruta'
              select(class=['ui', 'dropdown'] name="idruta")
                option(value=autobus[0].idruta selected hidden)= autobus[0].nombreRuta
                each ru in ruta
                  if autobus[0].idruta != ru.idruta
                    option(value=ru.idruta)= ru.nombreRuta

            div(class=['ui','field'])
              label(for='capacidad')= 'Capacidad'
              input(type='text' name="capacidad" placeholder='Capacidad' value=autobus[0].capacidad)
            div(class=['ui','field'])
              label(for='idconductor')= 'Conductor'
              select(class=['ui', 'dropdown'] name="idconductor")
                option(value=autobus[0].idconductor selected hidden)= autobus[0].nombreConductor
                each con in conductor
                  if autobus[0].idconductor != con.idconductor
                    option(value=con.idconductor)= con.nombreConductor

            div(class=['ui','field'])
              label(for='idmonitor')= 'Monitor'
              select(class=['ui', 'dropdown'] name="idmonitor")
                option(value=autobus[0].idmonitor selected hidden)= autobus[0].nombreMonitor
                each mon in monitor
                  if autobus[0].idmonitor != mon.idmonitor
                    option(value=mon.idmonitor)= mon.nombreMonitor

              input(type='hidden' name="id" value=autobus[0].id)
          button(class=['ui','button','basic','black'])= 'Modificar Autobús'
          div(class=['ui','message','error'])


  script(src='/javascripts/validacion.js')
```

Ilustración 77: Implementación - Vista modificar autobús

El código de la Ilustración 76 se va a ejecutar en el momento en el que pulsemos el botón modificar del autobús. Una vez pulsado se mostrará la vista “Modificar Autobús” cuyo código es el de la Ilustración 77. Se destaca que los campos de entrada de datos contendrán los valores por defecto del antiguo registro. Esto se consigue mediante el parámetro del *input*, *value=autobus[0].matricula*. Una vez se realicen los cambios deseados y se pulse el botón de “Modificar” se enviará el formulario con acción “/editar”, método POST y se ejecutará el código de la Ilustración 78.

```
postModificarAutobus : function(req, res, next){
  //recuperar autobus
  var autobus = {
    matricula : req.body.matricula,
    idruta : req.body.idruta,
    capacidad : req.body.capacidad,
    idconductor : req.body.idconductor,
    idmonitor : req.body.idmonitor
  };
  // console.log(autobus)
  var config = require('.././database/config');

  var db = mysql.createConnection(config);

  db.connect();
  db.query('UPDATE autobus SET ? WHERE ?',[autobus, {id : req.body.id}], function(err,rows,fields){
    if(err) throw err;
    db.end();
  });

  res.redirect('/autobus');
```

Ilustración 78: Implementación - Modificar autobús - Método POST

Mediante la variable “autobus” y los parámetros de “req.body”, obtenemos los valores que se han introducido en la vista de “Modificar Autobús”, se crea de nuevo una conexión con la base de datos mediante el comando “mysql.createConnection(config)” y “db.connect()” y se realiza la modificación utilizando la instrucción SQL *UPDATE SET*, pasándole la “id” del autobús que previamente recogimos.

Al finalizar, el usuario será redirigido al listado de autobuses con los nuevos cambios realizados.

5.2.7 Borrar elemento

La función de borrar elemento se ha realizado utilizando la técnica AJAX y su código es el siguiente.

```
a(class=['ui button basic icon '] href='#' data-inverted='' data-tooltip='Eliminar' data-position='top center' id='btn-eliminar')
  i(class=['trash icon red'])
```

Ilustración 79: Implementación - Botón eliminar autobús

La primera parte del código (Ilustración 79) representa el botón de eliminar. El dato a destacar es su “id” (btn-eliminar) la cual se utilizará en la llamada AJAX de la Ilustración 80

```
$(function(){

  //funcion ajax para eliminar autobús

  $('#tbl-autobus #btn-eliminar').click(function(e) {
    e.preventDefault();
    var elemento = $(this);
    var id = elemento.parent().parent().find('#id').text();

    var confirmar = confirm('¿Desea eliminar el autobus?')

    if(confirmar){
      $.ajax({
        url : '/eliminarautobus',
        method : 'post',
        data : {id : id},
        success : function(res){
            if(res.res){
              elemento.parent().parent().remove();
            }
        }
      });
    }

  });
```

Ilustración 80: Implementación - Eliminar autobús – AJAX

Haciendo uso del selector que nos proporciona JQuery, seleccionamos la "id" de la tabla autobús, junto con la id del botón "eliminar". Indicando que una vez se pulse, se procederá a la ejecución del resto de código, en el cual se recoge la id del autobús utilizando "elemento.parent().parent().find('#id').text()". Con esta id se procede a realizar la llamada AJAX la cual tiene los siguientes atributos:

- url: URL donde está escuchando nuestra función de eliminar autobús
- method: Método con el que se realizará la acción, en nuestro caso POST
- data: Valores que le pasamos a la llamada, en este caso estamos pasando la id del autobús que se desea eliminar.
- success: Indica que sucederá si la llamada devuelve *True*, en este caso se borrará el elemento de la vista del listado de autobuses.

El siguiente paso en el flujo de código es la llamada a nuestra función que escucha en la URL "/eliminarAutobus" (Ilustración 81).

```
eliminarAutobus : function(req, res, next){
    var id = req.body.id;
    var config = require('.././database/config');
    var db = mysql.createConnection(config);
    db.connect();
    var respuesta = {res: false};
    db.query('DELETE FROM autobus WHERE id = ?', id, function(err,rows,fields){
      if(err) throw err;

      db.end();
      respuesta.res = true;
      res.json(respuesta);
    });

},
```

Ilustración 81: Implementación - Eliminar Autobús - Método POST

En el código observamos como se recoge de nuevo la "id" del autobús, se realiza una conexión a la base de datos y mediante la instrucción SQL DELETE, se procede a eliminar el autobús que el usuario ha solicitado.

5.2.8 Subida y visualización de imágenes

La subida de imágenes en los parientes se ha conseguido utilizando la librería *Multer* de Node.js.

Primero, se define en el código como indica la ilustración 82.

```
var multer= require("multer");
var upload= multer({dest:"uploads/"});
```

Ilustración 82: Implementación - Subida de imágenes – Multer

Con la primera línea se define la librería, y con la segunda se especifica el fichero de destino donde se van a guardar las imágenes. En nuestro caso se va a llamar "uploads".

A continuación, se especifica lo que se va a subir en la ruta donde se va a utilizar esta funcionalidad (Ilustración 83).

```
router.post('/crearPariente', upload.single('foto'), controllers.parientecontroller.postNuevoPariente);

router.post('/editarPariente',upload.single('foto'), controllers.parientecontroller.postModificarPariente);
```

Ilustración 83: Implementación - Subida de imágenes – Rutas

Haciendo uso de la variable definida "upload" le especificamos que se va a subir una única foto, con id "foto" mediante la instrucción "upload.single('foto')"

Finalmente, en la función de crear pariente y editar pariente se recogerá el nombre de la imagen y se guardará en la base de datos. Este nombre se va a usar de manera que coincida con el que se ha guardado en el fichero "uploads" para que se utilice en el mismo pariente que se subió dicha imagen (Ilustración 84).

```
postNuevoPariente : function(req, res, next){

  var pariente = {
    nombre : req.body.nombre,
    foto : req.file.filename,
    idmatriculado : req.body.idmatriculado,
    email : req.body.email,
    telefono : req.body.telefono

  };

  var config = require('.././database/config');
  var db = mysql.createConnection(config);
  db.connect();
  db.query('INSERT INTO pariente SET ?', pariente, function(err,rows,fields){
    if(err) throw err;
    db.end();
  });
  res.render('pariente/nuevoPariente', {info : 'Pariente creado correctamente', matriculado : pariente.idmatriculado});
```

Ilustración 84: Implementación - Subida de imágenes - Método POST

Se destaca que el atributo foto es el nombre de la imagen, el cual mediante la instrucción SQL INSERT, es introducido como nuevo registro en la base de datos.

Hay que tener en cuenta que, para procesar la imagen, el formulario de la vista debe de contener esta línea:

enctype='multipart/form-data'

La cual permite enviar datos codificados.

5.2.9 Consulta de estado

Esta sección trata de la consulta de estado del alumno, la cual es la única que puede ser realizada por cualquier usuario.

```
extends default

block content
  div(class=['ui','menu'] style='background-color:#EDEDED;')

    a(class='item' href='/publica')

      span= 'Inicio'
  div(class='form-nuevomatriculado')
    form(class=['ui','form'] action='/publica_matriculado' method='post')

      div(class=['ui','container','grid','stackable'])
        div(class=['ten','wide','column'])
          div(class=['ui','field'])
            label(for='dni')= 'DNI Alumno'
            input(type='text' name="dni" placeholder='DNI')


          button(class=['ui','button','basic','black'])= 'Buscar Alumno'
```

Ilustración 85: Implementación - Consulta estado alumno - Introducir DNI

En primer lugar, es necesario introducir el DNI del alumno el cual se desea conocer el estado. El código de la vista se puede observar en la Ilustración 85.

Una vez pulsado el botón de buscar, mediante el método POST se realizará la acción /publica_matriculado definida en el formulario, ejecutando el siguiente código (Ilustración 86)

```
publica_matriculado : function(req, res, next){
  var dni = req.body.dni

  var config = require('.././database/config');
  var db = mysql.createConnection(config);
  db.connect();
  db.query('SELECT * from matriculado where dni = ?;', dni, function(err, rows, fields){
      if(err) throw err;
      var matriculado = rows;
  console.log(matriculado);
  db.end();
  res.render('publica_matriculado', {title : 'Autobús CP', matriculado:matriculado});

  });
```

Ilustración 86: Implementación - Consulta de estado - Método POST

El procedimiento es el mismo de siempre, se recoge el valor necesario para la búsqueda en la base de datos, en este caso el DNI, conectamos con la base de datos y realizamos una consulta que coincida con ese DNI.

Los resultados se pasarán por contexto a la vista “publica_matriculado”, la cual será mostrada automáticamente una vez realizada la búsqueda. En la Ilustración 87 vemos el código de esa vista. Se puede apreciar la tabla a mostrar, con el nombre completo y apellidos del alumno, así como el estado en el que se encuentra, que puede ser “Sin Registrar”, “Registrado” o “Incidencia”, dependiendo de si está en el autobús o no, o si ha habido algún problema.

En la parte final se observa la llamada al código de *Google Maps*, el cual muestra las coordenadas de la parada en la que se encuentra el alumno.

```
extends default

block content
  div(class=['ui','menu'] style='background-color:#EDEDED;')

    a(class='item' href='/publica')

      span= 'Inicio'
  div(class=['ui','container','grid','stackable'])
    div(class=['sixteen','wide','column'])
      table(class=['ui','table','celled'])
        thead
          tr

            th= 'Nombre'
            th= 'Nombre 2º'
            th= 'Apellido P'
            th= 'Apellido M'
            th= 'Dni'
            th= 'Estado'

        tbody
          each mat in matriculado
            tr

              td= mat.primer_nombre
              td= mat.segundo_nombre
              td= mat.apellido_paterno
              td= mat.apellido_materno
              td= mat.dni
              td= mat.estado



    div(id="map" style="width:350px;height:350px;background:yellow;float:right;display:inline-block; margin-top:5em;")

  script(src='/javascripts/googleMap.js')
  script(src='https://maps.googleapis.com/maps/api/js?key=AIzaSyC06gfUvlrhKBwtFx9j9w90BWIwFqZXWzY&callback=myMap')
```

Ilustración 87: Implementación - Consulta de estado - Resultado

5.2.10 Filtrado de elementos

Puede darse el caso que en algún momento el listado de los elementos sea tan grande que impida la búsqueda de alguno en concreto. Por este motivo se proporciona un método de filtro por diferentes campos que solucione este problema. Para desarrollarlo se ha utilizado una función en JavaScript con el evento “onkeyup”. El código se divide en tres partes y se utilizará de ejemplo el elemento autobús.

Primero, en la vista se ha creado el campo por el que se desea filtrar y se le ha asignado una función, que se ejecutará cada vez que el valor de ese campo cambie. (Ilustración 88).

```
input(type='text' name="filtro" onkeyup="myFunction()" placeholder='Filtrar por matricula' id="filtro")
```

Ilustración 88: Implementación - Filtro matrícula

Como se observa la función se llama “myFunction()” y ejecuta el siguiente código (Ilustración 89).

```
function myFunction() {
var input, filter, table, tr, td, i;
input = document.getElementById("filtro");

filter = input.value.toUpperCase();
table = document.getElementById("tbl-autobus");
tr = table.getElementsByTagName("tr");

for (i = 0; i < tr.length; i++) {
  td = tr[i].getElementsByTagName("td")[1];
  if (td) {
    if (td.innerHTML.toUpperCase().indexOf(filter) > -1) {
      tr[i].style.display = "";
    } else {
      tr[i].style.display = "none";
    }
  }
}
}
```

Ilustración 89: Implementación - Función filtro

Su funcionamiento consiste en seleccionar la tabla del listado y las filas, recorrerla mediante un bucle y ocultar aquellas en las que el valor escrito no coincida con el del atributo del elemento.

5.2.11 Notificación de correo electrónico

Con el fin de avisar a los parientes de los alumnos que han sufrido un atasco en una determinada ruta, se ha desarrollado una funcionalidad que permite enviar correos automatizados y personalizados pulsando un botón. Se ha utilizado la técnica AJAX para su desarrollo.

En primer lugar, en la vista del listado de rutas se ha añadido un botón de informar de atasco. El código se puede ver en la Ilustración 90.

```
a(class=['ui button basic icon '] href='#' data-inverted='' data-tooltip='Informar de atasco' data-position='top center' id='email')
  i(class=['mail center icon white'])
```

Ilustración 90: Implementación - Enviar correo – Botón

El campo a destacar es la id “email”, la cual se va a usar como selector mediante JQuery, y en cuanto se pulse el botón comenzará el flujo de envío de mail.

En la Ilustración 91 se aprecia como mediante AJAX se llama a la función “/enviarMail” junto con el método POST.

```
$('#tbl-ruta #email').click(function(e) {
  e.preventDefault();
  var elemento = $(this);
  var id = elemento.parent().parent().find('#id').text();


  var confirmar = confirm('¿Desea mandar un aviso de atasco a los padres de los alumnos?')

  if(confirmar){
    $.ajax({
      url : '/enviarMail',
      method : 'post',
      data : {idruta : id},
      success : function(res){
          if(res.res){
            alert("Mail mandando correctamente");
          }
      }
    });
  }

});
```

Ilustración 91: Implementación - Enviar mail – AJAX

```
enviarMail : function(req, res, next){
    var id = req.body.idruta;
    // console.log(id)
    // console.log(req.body);
    var config = require('.././database/config');
    var db = mysql.createConnection(config);
    db.connect();
    var respuesta = {res: false};
    db.query('SELECT p.email, r.nombreRuta, m.primer_nombre FROM ruta r, matriculado m,\
     pariente p WHERE r.idruta = ? AND r.idruta = m.idruta AND m.idmatriculado = p.idmatriculado', id, function(err,rows,fields){
      if(err) throw err;
      db.end();
      var nodemailer = require('nodemailer');

      var transporter = nodemailer.createTransport({
        service: 'Gmail',
        auth: {
          user: 'aplicacionautobustfg@gmail.com',
          pass: 'manolo17'
        }
      });
      for (row in rows) {


      var mailOptions = {
        from: 'AplicacionAutobus',
        to: rows[row].email,
        subject: 'Atasco en la ruta '+rows[row].nombreRuta,
        text: 'Hola, le informo de que el autobús que lleva al alumno '+rows[row].primer_nombre+' se ha retrasado debido a un atasco, disculpe las molestias.'
      };
      transporter.sendMail(mailOptions, function(error, info){
        if (error){
          console.log(error);
          res.send(500, err.message);
        } else {
          console.log("Email Enviado");
          res.status(200).jsonp(req.body);
        }
      });

      }
      respuesta.res = true;
      res.json(respuesta);
```

Ilustración 92: Implentación - Enviar mail - Método POST

Por último, la Ilustración 92 enseña el proceso que realiza la parte servidor para el envío del mail. Se utiliza la librería *nodemailer*, y una vez declarada esta variable, se crea un objeto *transporter* mediante el método “nodemailer.createTransport”, el cual contiene los siguientes atributos:

- Service: El servicio de mensajería que se usa para enviar el correo, en nuestro caso es Gmail.
- User: El usuario que envía el correo.
- Password: Contraseña del usuario.

Después, se crea también la variable mailOptions, con estos atributos:

- From: Desde que cuenta se envía el correo.
- To: A que cuenta se envía el correo. En nuestro caso se enviará un correo a cada pariente que tenga un alumno en la ruta del atasco.
- Subject: Asunto del mensaje. Personalizado con el nombre de la ruta que ha sufrido el atasco.
- Text: Cuerpo del mensaje. Personalizado con el nombre del alumno a cuyo pariente se le envía el correo.

Para finalizar, se utiliza la línea "transporter.sendMail(mailOptions)", la cual manda tantos correos electrónicos(con las opciones que se han definido) como parientes que tengan un alumno en esa ruta.

5.2.12 Notificación de Telegram

Otra funcionalidad disponible es la de informar a los parientes del estado actual del alumno.

La realización de las pruebas se ha hecho utilizando Docker y una plantilla de Ubuntu 14.04, sin embargo, en la práctica se haría uso de una Raspberry Pi [2] o de un servidor remoto. Los pasos para la configuración del entorno han sido los siguientes:

- Una vez conectados al contenedor Docker o a la Raspberry Pi, se ha actualizado el sistema operativo y sus paquetes mediante el uso de las líneas de comando **"sudo apt-get update y sudo apt-get upgrade".**
- Se ha descargado el paquete TG mediante "**git clone –recursive https://github.com/vysheng/tg.git && cd tg**"
- Se han instalado todas las librerías necesarias utilizando esta línea de comando: **"sudo apt-get install libreadline-dev libconfig-dev libssl-dev lua5.2 liblua5.2-dev".**
- Una vez instaladas, se ejecuta el comando **"/configure"** y "**make***"* para compilar el programa.

Con el entorno y la estructura construida, con el fin de automatizar el envío de mensajes, es necesario formar una conexión entre el servidor donde está la aplicación y la Raspberry Pi.

Se ha utilizado el siguiente código para la realización de dicha conexión:

```
enviarNotificacion : function(req, res, next){
    var id = req.body.idmatriculado;

    var config = require('.././database/config');
    var db = mysql.createConnection(config);
    db.connect();
    var respuesta = {res: false};
    db.query('SELECT * FROM matriculado WHERE idmatriculado = ?', id, function(err,rows,fields){
      if(err) throw err;
      var matriculado = rows;
      db.query('SELECT * from pariente where idmatriculado = ?;', id, function(err, rows, fields){
          if(err) throw err;
          pariente = rows;
      db.end();


      var rexec = require('remote-exec');


      var connection_options = {
          port: 22,
          username: 'root',
          password: 'Dx1234!!'

      };

      var hosts = [
          '81.202.177.112'
      ];

      var cmds = [
          './script_telegram.sh '+ pariente[0].nombre + ' "' + "Estado de " + matriculado[0].primer_nombre + ": "
          + matriculado[0].estado + "." + " Compruebe nuestra aplicación para más información" + '"'
      ];

      rexec(hosts, cmds, connection_options, function(err){
          if (err) {
              console.log(err);
          } else {
              console.log('Envío Correcto');
          }
      });

      respuesta.res = true;
      res.json(respuesta);
```

Ilustración 93: Implementación - Notificación telegram - Conexión entre servidores

En la Ilustración 93 se observa la función del controlador “enviarNotificación” en la cual, la primera parte consiste en la búsqueda del alumno. Guardamos sus atributos en la variable “matriculado” así como información del pariente en la variable “pariente”. A continuación, utilizando la librería “remote-exec” realizamos una conexión con la Raspberry Pi.

Las opciones de la conexión son las siguientes:

- Port: 22 – Es el puerto por defecto usado para una conexión SSH.
- Username: root – El usuario utilizado para la conexión SSH
- Password: Contraseña para conectarse correctamente.

La variable “hosts” contiene las direcciones IP a las que se va a realizar dicha conexión.

La variable “cmds” contiene el comando que va a ser llamado en la Raspberry Pi, en nuestro caso es un *script* cuyos parámetros de lanzamiento serán el nombre del pariente al que se va a notificar y el mensaje que se va a enviar, proporcionando los datos del estado del alumno.

El contenido del *script* es el siguiente:

```
#!/bin/bash
to=$1
msg=$2
tgpath=/home/pi/tg
cd ${tgpath}
(echo "msg $to $msg"; echo "safe_quit") | ${tgpath}/bin/telegram-cli -k tg-server.pub -W
```

Ilustración 94: Implementación - Script de notificación de Telegram

En la Ilustración 94 podemos comprobar las instrucciones que se van a lanzar en la Raspberry Pi. Guardamos en variables el destinatario, el mensaje y la ruta donde está el programa de código abierto instalado. Por último, haciendo uso del comando “/bin/telegram-cli –k tg-server.pub –W” publicamos el mensaje al destinatario que utilizamos como parámetro, informando del estado del alumno.

El mensaje enviado tanto en el móvil como en la consola es el siguiente:

```
> msg AplicacionTFG Estado de Alfonso: 'Incidencia'. Compruebe nuestra aplicacion para mas informacion
[22:30] AplicacionTFG <<< Estado de Alfonso: 'Incidencia'. Compruebe nuestra aplicacion para mas informacion
```

Ilustración 95: Implementación - Mensaje de Telegram enviado por consola

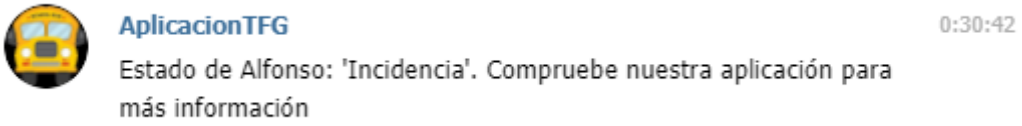


Ilustración 96 - Implementación - Mensaje de Telegram enviado al dispositivo

En las Ilustraciones 95 y 96 comprobamos que se ha enviado un mensaje personalizado al pariente del alumno con su estado actual.

6. Validación y pruebas

En esta sección de la memoria se pretende enseñar las pruebas que se han realizado para comprobar que todo el desarrollo anteriormente citado funciona perfectamente.

En primer lugar, un punto fundamental en la aplicación es que todas las funciones CRUD guardaran correctamente los valores en la base de datos y no provocaran errores. Para resolver la problemática, uno de los principales métodos que se ha utilizado ha sido la depuración mediante *logs*. JavaScript ofrece la función "console.log()", la cual permite usar cualquier variable y conocer su valor actual.

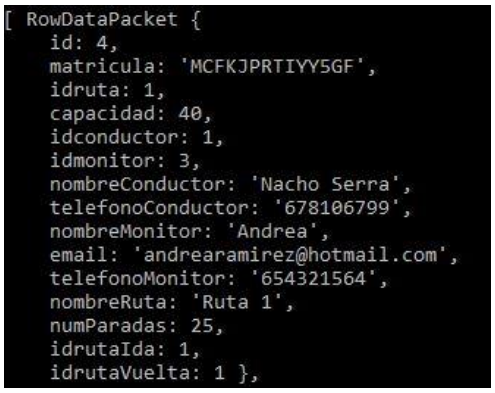


Ilustración 97: Validación y pruebas - Ejemplo de log

Por ejemplo, en la Ilustración 97 estamos viendo el primer elemento del listado de autobuses y todos sus atributos. Cuando la aplicación finalizaba con un error, se utilizaba esta técnica para saber de dónde provenía. Una vez solucionado se comprobaba que el valor de estos registros fuera el mismo que el guardado en la base de datos, comprobando que ni la parte de vista ni la de servidor produjera un error.

Para un mayor control de los posibles errores se ha hecho uso de la cláusula *try catch*, la cual impide que un error de una nueva casuística bloquee la aplicación completamente.

En la parte de vista se ha utilizado el siguiente código para la validación de datos.

```
$(function(){
  $('.form-nuevoautobus form').form({
      matricula : {
        identifier : 'matricula',
        rules : [
          {
            type : 'empty',
            prompt : 'Por favor ingrese una matrícula'
          }
        ]
      },
      idruta : {
        identifier : 'idruta',
        rules : [
          {
            type : 'empty',
            prompt : 'Por favor ingrese una ruta'
          }
        ]
      },
      capacidad : {
        identifier : 'capacidad',
        rules : [
          {
            type : 'empty',
            prompt : 'Por favor ingrese una capacidad'

          },
          {
            type: 'integer',
            prompt: 'La capacidad debe de ser número entero'
          }

        ]
```

Ilustración 98: Validación y pruebas - Ejemplo validación vista

La Ilustración 98 muestra un ejemplo que se ha usado para la validación de los campos del autobús. Donde no permite campos vacíos ni campos incorrectos, como por ejemplo que la capacidad esté formada por letras en vez de números enteros.

Un objetivo principal ha sido que la interfaz visual sea altamente usable, al ser una aplicación que va a ser usada por alguien sin conocimientos de informática, tiene que ser intuitiva y funcional en cualquier dispositivo. Se han realizado diferentes pruebas con personas expertas e inexpertas que pudieran cometer fallos en los flujos de creación, modificación, envío de mails… para que, de esta forma pudieran preguntar dudas u ofrecer sugerencias que mejoraran este aspecto de la aplicación.

7. Conclusiones

Finalizado el proyecto, llegamos a varias conclusiones relacionadas con el desarrollo de la aplicación.

En primer lugar, cabe destacar el aprendizaje que ha proporcionado el proyecto. Partiendo de cero en casi todos los aspectos de la implementación, Node.js ha demostrado ser muy potente y ha proporcionado una visión diferente relacionada con la parte del servidor. El motor de plantillas Jade también ha sido una sorpresa, permitiendo maquetar rápidamente la estructura de las vistas.

Tal y como ha quedado la aplicación, ofrece un control completo de la parte de gestión de autobuses, permitiendo modificar cualquier elemento que se desee.

También cumple la consulta de estado de los alumnos, donde los parientes pueden conocer si los hijos están dentro del autobús o no, o si ha habido alguna incidencia. Además, el administrador también puede informar a los padres pulsando un simple botón.

El apartado que se puede mejorar en un futuro es el sistema de inicio de sesión, en el cual, se podría añadir un sistema de registro de cuentas.

A nivel personal, la realización del trabajo ha supuesto un incremento a nivel de conceptos y de programación, y sin duda se han adquirido nuevas competencias centradas en el desarrollo de aplicaciones web.

8. Bibliografía

[1] Garrido, S. (2015). *Diseño de base de datos: Un enfoque práctico*. España: Amazon Media EU S.à.r.l

[2] Waher, P. (2015). *Learning Internet of Things*. Birmingham: Packt Publishing Ltd.

[3] Kiessling, M. (2012). *The Node Beginner Book*. Cologne, Germany: Leanpub

[4] Kiessling, M. (2017). *The Node Craftsman Book*. Cologne, Germany: Leanpub

[5] Hierror, Pfeiffer, A, Jeremie. Recuperado de https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

[6] Rouse, M. (2015). *Framework*. Recuperado de http://whatis.techtarget.com/definition/framework

[7] *The Apache Software Foundation.* <https://www.apache.org/> [Consulta: 5 de junio de 2017]

[8] *Javascript.* <https://www.javascript.com/> [Consulta: 5 de junio de 2017]

[9] *Node JS.* <https://nodejs.org/es/> [Consulta: 10 de junio de 2017]

[10] *The World's Largest Web Developer Site: SQL Tutorial.* <https://www.w3schools.com/SQL/deFault.asp> [Consulta: 12 de junio de 2017]

[11] *The World's Largest Web Developer Site: CSS Tutorial.* <https://www.w3schools.com/css/> [Consulta: 12 de junio de 2017]

[12] *Semantic UI: User Interface is the language of the Web.* <https://semantic-ui.com/> [Consulta: 13 de junio de 2017]

[13] *Learn Jade* <http://learnjade.com/> [Consulta: 14 de junio de 2017]

[14] *The World's Largest Web Developer Site: HTML 5 Tutorial.* <https://www.w3schools.com/html/default.asp> Consulta: 17 de junio de 2017]

[15] *The World's Largest Web Developer Site: Ajax Introduction.* <https://www.w3schools.com/xml/ajax_intro.asp> [Consulta: 3 de julio de 2017]

[16] *MySQL : MySQL Workbench*. <https://www.mysql.com/products/workbench/> [Consulta: 5 de julio de 2017]

[17] *jQuery*. <http://jquery.com/> [Consulta: 5 de julio de 2017]