Niall Hughes

B00634752

Enterprise Computing

Assignment 2

# Table of Contents

[Table of Contents i](#_Toc468429109)

[Question 1 – ASP.NET MVC Entity Framework 1](#_Toc468429110)

[MotCentre C# Class Model 1](#_Toc468429111)

[Design Rational 1](#_Toc468429112)

[Appointment C# Class Model 2](#_Toc468429113)

[Design Rational 3](#_Toc468429114)

[MotCentreRepository Code 3](#_Toc468429115)

[AppointmentRepository Code 6](#_Toc468429116)

[Design Rational 7](#_Toc468429117)

[MotCentreController 7](#_Toc468429118)

[Design Rational 12](#_Toc468429119)

[AppointmentController 13](#_Toc468429120)

[Design Rational 17](#_Toc468429121)

[Mot Centre Opening Hours 18](#_Toc468429122)

[MotCentre Demo 19](#_Toc468429123)

[Create a MOTCentre 19](#_Toc468429124)

[Edit a MOTCentre 22](#_Toc468429125)

[Details of a MOTCentre 24](#_Toc468429126)

[Deleting a MOTCentre 25](#_Toc468429127)

[List of MOTCentres 26](#_Toc468429128)

[Appointment Demo 27](#_Toc468429129)

[Creating an Appointment 27](#_Toc468429130)

[Edit an Appointment 30](#_Toc468429131)

[Detailing an Appointment 32](#_Toc468429132)

[Deleting an Appointment 32](#_Toc468429133)

[List of all Appointments 33](#_Toc468429134)

[List of a Centre’s Appointments 33](#_Toc468429135)

# Question 1 – ASP.NET MVC Entity Framework

## MotCentre C# Class Model

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

namespace Assignment2MOT

{

public partial class MOTCentre

{

/\*

\* This class is never used to take data for the user and so therefore no data validation is applied to the fields.

\*/

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public MOTCentre()

{

CentreTimes = new HashSet<CentreTime>();

VechAppoints = new HashSet<VechAppoint>();

}

[Key]

public int CentreId { get; set; }

[Required]

[Display(Name = "Centre Name")]

public string CentreName { get; set; }

[Required]

[Display(Name = "Centre Address")]

public string CentreAddress { get; set; }

[Required]

[Display(Name = "Telephone Number")]

[DisplayFormat(DataFormatString = "{0:0## #### ####}")]

public long CentreTeleNo { get; set; }

[Required]

[Display(Name = "Centre County")]

public string CentreCounty { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<CentreTime> CentreTimes { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<VechAppoint> VechAppoints { get; set; }

}

}

### Design Rational

In the MOTCentre class I’ve assigned each data fields to an appropriate type. Name, Address and County are recorded as strings as they are strings of characters. Telephone Number is stored as a long integer type. This is because a regular int is not long enough to hold a 10 digit number (assuming we ignore the leading zero). This restricts the user to only being able to enter numeric characters. When displaying the Centre’s Phone number, a DisplayFormat is set so that it always displays with a leading zero and has spaces inserted for better readability.

The MOTCentre class has links to both the VechAppoint class and the CentreTimes class (which is used for storing information about a specific classes opening hours).

Each property is given a Display Name which is used when Razor requests a label for the field. They are also tagged with the Required annotation however since this class is never used for gathering data they are never used. The validation of inputted data will be discussed in the creation and edit of MOT Centres section.

## Appointment C# Class Model

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

namespace Assignment2MOT

{

[Table("VechAppoint")]

public partial class VechAppoint

{

/\*

\* This class is never used to take data for the user and so therefore no data validation is applied to the fields.

\*/

[Key]

public int VechAppointId { get; set; }

[Required(ErrorMessage = "Vehicle Registration is a required field")]

[StringLength(7)]

[Display(Name = "Vehicle Registration")]

public string VechRegNo { get; set; }

[Required(ErrorMessage = "Vehicle Owner is a required field")]

[Display(Name = "Vehicle Owner")]

public string VechOwner { get; set; }

[Required(ErrorMessage = "Appointment Time is a required field")]

[Display(Name = "Appointment Time:")]

[DisplayFormat(DataFormatString = "{0:ddd dd MMM yyyy a\\t HH:mm}")]

public DateTime VechAppointTime { get; set; }

[Required(ErrorMessage = "MOT Centre is a required field")]

[Display(Name = "MOT Centre")]

public int MOTCentresCentreId { get; set; }

public virtual MOTCentre MOTCentre { get; set; }

}

}

### Design Rational

Much like the MOTCentre class, the VechAppoint class has little to no input validation as it is never used to gather data. Instead it handles the format of displayed information. Each propery has a Display Name for their label and the VechAppointTime has its DisplayFormat set to display the Date and Time of the appointment using a custom short date format. The letter t needed to be escaped twice due to it representing a specifier and the tab character.

This class has a link to the MOTCentre class which can be used to access the variables of it’s associated MOTCentre.

## MotCentreRepository Code

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Assignment2MOT.Models.Repositories

{

class MOTCentreRepository : IMOTCentresRepository

{

private MOTContext db = null;

// MOTCentreRepository Constructor

public MOTCentreRepository()

{

this.db = new MOTContext();

}

// MOTCentreRepository Constructor

public MOTCentreRepository(MOTContext db)

{

this.db = db;

}

// Delete a MOT Centre, delete's all times associated with it too

public void Delete(int id)

{

MOTCentre existing = db.MOTCentres.Find(id);

foreach( var time in GetTimes(existing.CentreId))

{

db.CentreTimes.Remove(time);

}

db.MOTCentres.Remove(existing);

}

// Insert a MOT Centre

public void Insert(MOTCentre obj)

{

db.MOTCentres.Add(obj);

}

// Save changes

public void Save()

{

db.SaveChanges();

}

// Select all MOT Centres

public IEnumerable<MOTCentre> SelectAll()

{

return db.MOTCentres.OrderBy(a => a.CentreName).Include(c => c.CentreTimes).ToList();

}

// Select MOT Centre by id

public MOTCentre SelectByID(int id)

{

return db.MOTCentres.Find(id);

}

/\*

// Select MOT Centre by name

public MOTCentre SelectByName(string name)

{

return db.MOTCentres.Where(a => String.Compare(a.CentreName, name) == 0).First();

}

\*/

// Update a MOT Centre

public void Update(MOTCentre obj)

{

MOTCentre centre = SelectByID(obj.CentreId);

centre.CentreName = obj.CentreName;

centre.CentreAddress = obj.CentreAddress;

centre.CentreCounty = obj.CentreCounty;

centre.CentreTeleNo = obj.CentreTeleNo;

}

// Update a MOT Centre's opening hours

public void UpdateTime(CentreTime obj)

{

IEnumerable<CentreTime> time = db.CentreTimes.Where(t => t.MOTCentresCentreId == obj.MOTCentresCentreId && t.DayOfTheWeek == obj.DayOfTheWeek);

//if time == 00 delete record if exists

if (obj.OpeningTime == obj.ClosingTime)

{

if (time.Any())

{

db.CentreTimes.Remove(time.First());

}

}

// Else we add the time if it doesn't exist or update it if it does

else

{

if (!time.Any())

{

db.CentreTimes.Add(obj);

}

else

{

time.First().OpeningTime = obj.OpeningTime;

time.First().ClosingTime = obj.ClosingTime;

}

}

}

// Get opening hours of a MOT Centre

public IEnumerable<CentreTime> GetTimes(int id)

{

return db.CentreTimes.Where(a => a.MOTCentresCentreId == id).ToList();

}

// Insert a Centre Time

public void InsertTime(CentreTime obj)

{

db.CentreTimes.Add(obj);

}

}

}

The MOTCentreRepository acts as the interface to the SQL database for the MOTCentre table and the CentreTimes table as these too are very closely related. A centre cannot be created without an opening time so therefore a MOTCentre entry cannot be created without creating at least one or more CentreTime entries.

For the most part the accessing of both tables is kept separate, the exception being the delete and SelectAll() methods. The SelectAll() method requires the CentreTimes as it need to display the associated opening hours of a Mot Centre in the same view. The delete method accesses the CentreTimes table as when deleting a MOT Centre naturally it will no longer have any opening hours and so all associated CentreTime entries are also deleted.

The Update() method updates the changes made to an MOT Centre. The UpdateTime() method finds the day of the week that the Centre is now opening on. Should one not exist, a new CentreTime will be created. If the new opening hours are equal to each other it is assumed the centre is closed for that day and the record is deleted. And if the record does exist the hours are updated with the new times.

A GetTimes helper method is also supplied for returning the opening hours of a Centre by inputting the MOT Centre ID. This is useful for finding the specific times of one centre.

## AppointmentRepository Code

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Web;

namespace Assignment2MOT.Models.Repositories

{

public class VechAppointRepository : IVechAppointRepository

{

private MOTContext db = null;

// VechAppointRepository Constructory

public VechAppointRepository()

{

this.db = new MOTContext();

}

// VechAppointRepository Constructory

public VechAppointRepository(MOTContext db)

{

this.db = db;

}

// Delete an appointment

public void Delete(int id)

{

VechAppoint existing = db.VechAppoints.Find(id);

db.VechAppoints.Remove(existing);

}

// Insert an appointment

public void Insert(VechAppoint obj)

{

db.VechAppoints.Add(obj);

}

// Save all changes

public void Save()

{

db.SaveChanges();

}

// Get all appointments

public IEnumerable<VechAppoint> SelectAllAppointments()

{

return db.VechAppoints.OrderBy(a => a.MOTCentresCentreId).ThenBy(b => b.VechAppointTime).ToList();

}

// Get Appointments for Centre matching ID

public IEnumerable<VechAppoint> SelectCentreAppointments(int id)

{

return db.VechAppoints.Where(a => a.MOTCentresCentreId == id).OrderBy(b => b.VechAppointTime).ToList();

}

// Get Appointment by ID

public VechAppoint SelectByID(int id)

{

return db.VechAppoints.Find(id);

}

// Update an appointment

public void Update(VechAppoint obj)

{

VechAppoint va = SelectByID(obj.VechAppointId);

va.VechAppointTime = obj.VechAppointTime;

va.VechOwner = obj.VechOwner;

va.VechRegNo = obj.VechRegNo;

va.MOTCentresCentreId = obj.MOTCentresCentreId;

}

public IEnumerable<string> SelectAllRegs(int id)

{

return db.VechAppoints.Where(m => m.VechAppointId != id).Select(m => m.VechRegNo).ToList();

}

}

}

### Design Rational

The VechAppointRepository acts as the interface for the VechAppoint table of the SQL database. This allows actions such as viewing, creating, editing and deleting the records of the database. Create, Delete, Insert, Save and SelectByID all function very simply.

SelectAllAppointments() sorts the list of appointments first by the MOT Centre then by the date. This provides the clearest and easiest format for users to be able to view the appointments. The SelectCentreAppointments() operates under the same principle but is able to filter the appointments by centre when given a Centre ID.

The Update() method finds a centre and updates it according to the new details. Finally SelectAllRegs() is used to get a list of Licence Plate numbers which helps to prevent two cars being registered for different appointments.

## MotCentreController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Assignment2MOT.Models.Repositories;

using Assignment2MOT.Models;

using System.Text.RegularExpressions;

namespace Assignment2MOT.Controllers

{

public class MOTCentreController : Controller

{

private MOTCentreRepository repository = null;

public List<SelectListItem> list = new List<SelectListItem>{

new SelectListItem{ Text="00:00", Value = "0" },

new SelectListItem{ Text="01:00", Value = "1" },

new SelectListItem{ Text="02:00", Value = "2" },

new SelectListItem{ Text="03:00", Value = "3" },

new SelectListItem{ Text="04:00", Value = "4" },

new SelectListItem{ Text="05:00", Value = "5" },

new SelectListItem{ Text="06:00", Value = "6" },

new SelectListItem{ Text="07:00", Value = "7" },

new SelectListItem{ Text="08:00", Value = "8" },

new SelectListItem{ Text="09:00", Value = "9" },

new SelectListItem{ Text="10:00", Value = "10" },

new SelectListItem{ Text="11:00", Value = "11" },

new SelectListItem{ Text="12:00", Value = "12" },

new SelectListItem{ Text="13:00", Value = "13" },

new SelectListItem{ Text="14:00", Value = "14" },

new SelectListItem{ Text="15:00", Value = "15" },

new SelectListItem{ Text="16:00", Value = "16" },

new SelectListItem{ Text="17:00", Value = "17" },

new SelectListItem{ Text="18:00", Value = "18" },

new SelectListItem{ Text="19:00", Value = "19" },

new SelectListItem{ Text="20:00", Value = "20" },

new SelectListItem{ Text="21:00", Value = "21" },

new SelectListItem{ Text="22:00", Value = "22" },

new SelectListItem{ Text="23:00", Value = "23" },

};

// Constructor

public MOTCentreController()

{

this.repository = new MOTCentreRepository();

}

// List of Centres

[HttpGet]

public ActionResult Index()

{

List<MOTCentre> model = (List<MOTCentre>)repository.SelectAll();

return View(model);

}

// Centre Details

[HttpGet]

public ActionResult Details(int id)

{

MOTCentre existing = repository.SelectByID(id);

if (existing == null)

{

return new HttpNotFoundResult("Invalid Centre ID");

}

return View(existing);

}

// Centre Create - display the view

[HttpGet]

public ActionResult Create()

{

ViewData["TimeList"] = list;

return View();

}

// Centre Create - get the result and process

[HttpPost]

public ActionResult Create(MotCentreTimes obj)

{

ViewData["TimeList"] = list;

if (ModelState.IsValid)

{ // check valid state and times

if (checkTimes(obj))

{

MOTCentre centre = convertMCTtoMC(obj);

repository.Insert(centre);

repository.Save();

for (int i = 0; i < obj.times.Count; i++)

{

if (obj.times[i].OpeningTime.TotalDays != obj.times[i].ClosingTime.TotalDays)

{

CentreTime time = new CentreTime { DayOfTheWeek = i, OpeningTime = TimeSpan.FromHours(obj.times[i].OpeningTime.TotalDays), ClosingTime = TimeSpan.FromHours(obj.times[i].ClosingTime.TotalDays), MOTCentresCentreId = centre.CentreId };

repository.InsertTime(time);

}

}

repository.Save();

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError("Time", "You have entered invalid opening hours. Centre must be oen for at least one day. A Centre also cannot Open after it Closes.");

return View(obj);

}

}

else // not valid so redisplay

{

return View(obj);

}

}

// Edit Centre - send the view

[HttpGet, ActionName("Edit")]

public ActionResult ConfirmEdit(int id)

{

ViewData["TimeList"] = list;

MOTCentre existing = repository.SelectByID(id);

if (existing == null)

{

return new HttpNotFoundResult("Invalid Centre ID");

}

MotCentreTimes centreTime = convertMCtoMCT(existing);

return View(centreTime);

}

// Edit Centre - handle the return

[HttpPost]

public ActionResult Edit(MotCentreTimes obj)

{

ViewData["TimeList"] = list;

if (ModelState.IsValid)

{ // check valid state

if (checkTimes(obj))

{

MOTCentre centre = convertMCTtoMC(obj);

repository.Update(centre);

repository.Save();

for (int i = 0; i < obj.times.Count; i++)

{

CentreTime time = new CentreTime { DayOfTheWeek = i, OpeningTime = TimeSpan.FromHours(obj.times[i].OpeningTime.TotalDays), ClosingTime = TimeSpan.FromHours(obj.times[i].ClosingTime.TotalDays), MOTCentresCentreId = centre.CentreId };

repository.UpdateTime(time);

}

repository.Save();

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError("Time", "You have entered invalid opening hours. Centre must be oen for at least one day. A Centre also cannot Open after it Closes.");

return View(obj);

}

}

else // not valid so redisplay

{

return View(obj);

}

}

// Delete centre - send the view

[HttpGet, ActionName("Delete")]

public ActionResult ConfirmDelete(int id)

{

MOTCentre existing = repository.SelectByID(id);

if (existing == null)

{

return new HttpNotFoundResult("Invalid Centre ID");

}

return View(existing);

}

// Delete centre - handle the return

[HttpPost]

public ActionResult Delete(int id)

{

repository.Delete(id);

repository.Save();

return RedirectToAction("Index");

}

/\*

\* Checks if the times given are valid.

\* i.e. not equal to each other.

\* doesn't close before it opens.

\*/

private bool checkTimes(MotCentreTimes Centre)

{

for (int i = 0; i < Centre.times.Count; i++)

{

if (TimeSpan.Compare(Centre.times[i].OpeningTime, Centre.times[i].ClosingTime) == 1)

{

return false;

}

}

for (int i = 0; i < Centre.times.Count; i++)

{

if (Centre.times[i].OpeningTime != Centre.times[i].ClosingTime)

{

return true;

}

}

return false;

}

// Conversion for MotCentreTimes to MOTCentre

private MOTCentre convertMCTtoMC(MotCentreTimes centreTime)

{

MOTCentre centre = new MOTCentre { CentreId = centreTime.CentreId, CentreName = centreTime.CentreName, CentreCounty = centreTime.CentreCounty, CentreTeleNo = long.Parse(Regex.Replace(centreTime.CentreTeleNo, @"\s+", "")) };

if (String.IsNullOrEmpty(centreTime.CentreAddressLn2))

{

centre.CentreAddress = centreTime.CentreAddressLn1 + "," + centreTime.CentreCounty + "," + centreTime.CentrePostcode;

}

else

{

centre.CentreAddress = centreTime.CentreAddressLn1 + "," + centreTime.CentreAddressLn2 + "," + centreTime.CentreCounty + "," + centreTime.CentrePostcode;

}

return centre;

}

// Conversion for MOTCentre to MotCentreTimes

private MotCentreTimes convertMCtoMCT(MOTCentre centre)

{

MotCentreTimes centreTime = new MotCentreTimes { CentreId = centre.CentreId, CentreName = centre.CentreName, CentreCounty = centre.CentreCounty, CentreTeleNo = ("0" + centre.CentreTeleNo.ToString()) };

string[] address = centre.CentreAddress.Split(',');

centreTime.CentreAddressLn1 = address[0];

if (address.Count() < 4)

{

centreTime.CentrePostcode = address[2];

}

else

{

centreTime.CentreAddressLn2 = address[1];

centreTime.CentrePostcode = address[3];

}

centreTime.times = new List<MotCentreTimes.ct>();

for (int i = 0; i <= 6; i++)

{

centreTime.times.Add(new MotCentreTimes.ct());

}

foreach (var time in repository.GetTimes(centreTime.CentreId))

{

centreTime.times[(time.DayOfTheWeek)] = new MotCentreTimes.ct { OpeningTime = time.OpeningTime, ClosingTime = time.ClosingTime };

}

return centreTime;

}

}

}

### Design Rational

One of the key innovations for the MotCentreController is the use of two classes to handle actions relating to a MOT Centre. This was done to allow greater amounts of validation control over each piece of data entered into the system. The MotCentreTimes class is used to get information from the user and has rigid validation rules applied to ensure the data is suitable to be returned to the database. The standard pipeline for data entry is as follows:

This functionality greatly reduces the possibility of an error when entering data and ensures all data stored is formatted appropriately.

For any views that require a specific entry such as the Details or Delete view a check is performed to ensure that the given ID belongs to an existing database member. If not a 404 Not found error is returned to the screen.

To allow for the selection of opening hours, a list of times is passed to the View using the ViewData. This list called “TimeList” is then used to populate each field of the Opening hours’ dropdown boxes for both the create and edit views.

When a centre to be created is returned it is checked to see if it is valid and then checked if the hours entered are valid. Should either of these fail appropriate messages are returned to the user through the View. Upon passing the data is converted and saved to the database. The Opening hours’ data is also extracted and saved. The function the returns the user to the page listing the Centres.

When editing a Centre, a check is first performed to ensure it exists then the centre is converted to a MotCentreTimes structure to use for validation. On return the data undergoes the same validation from the create method and the database is then updated.

Some helper methods are used such as conversion methods and a method used to validate the opening hours’. These are useful as they reduce the code in each of the HTTP functions and help keep unrelated functions separate.

## AppointmentController

using Assignment2MOT.Models.Repositories;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using Assignment2MOT.Models;

using System.Text.RegularExpressions;

namespace Assignment2MOT.Controllers

{

public class VechAppointController : Controller

{

private VechAppointRepository AppointRepository = null;

private MOTCentreRepository MOTRepository = null;

// Constructor for VechAppointController

public VechAppointController()

{

this.AppointRepository = new VechAppointRepository();

this.MOTRepository = new MOTCentreRepository();

}

// Constructor for VechAppointController for test purposes

public VechAppointController(VechAppointRepository repository)

{

this.AppointRepository = repository;

this.MOTRepository = new MOTCentreRepository();

}

// List the appointments, parameter is optional to display appointments relating to a specified centre

[HttpGet]

public ActionResult Index(int id = -1)

{

if (id >= 0)

{

List<VechAppoint> centreAppoints = (List<VechAppoint>)AppointRepository.SelectCentreAppointments(id);

return View(centreAppoints);

}

List<VechAppoint> model = (List<VechAppoint>)AppointRepository.SelectAllAppointments();

return View(model);

}

// Details an appointment

[HttpGet]

public ActionResult Details(int id)

{

VechAppoint existing = AppointRepository.SelectByID(id);

if (existing == null)

{

return new HttpNotFoundResult("Invalid Centre ID");

}

return View(existing);

}

// Creates an appointment

[HttpGet]

public ActionResult Create()

{

VechAppointViewModel viewModel = new VechAppointViewModel

{

MOTCentres = MOTRepository.SelectAll()

};

return View(viewModel);

}

// Creates an appointment - handles the response back

[HttpPost]

public ActionResult Create(VechAppointViewModel obj)

{

if (ModelState.IsValid)

{ // check valid state and time

if (isTimeValid(obj.VechAppointTime, obj.VechAppointDate.Value, obj.MOTCentresCentreId))

{

if (isRegValid(obj.VechRegNo))

{

VechAppoint newAppoint = convVAVMtoVA(obj);

AppointRepository.Insert(newAppoint);

AppointRepository.Save();

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError("Reg", "There is already a Vehicle with that Registration Number.");

obj.MOTCentres = MOTRepository.SelectAll();

return View(obj);

}

}

else

{

ModelState.AddModelError("Time", "Sorry but the Centre you have selected is not open at this time. Please refer to the MOT Centres Table and reselect a time.");

obj.MOTCentres = MOTRepository.SelectAll();

return View(obj);

}

}

else // not valid so redisplay

{

obj.MOTCentres = MOTRepository.SelectAll();

return View(obj);

}

}

// Edits an appointment

[HttpGet, ActionName("Edit")]

public ActionResult ConfirmEdit(int id)

{

VechAppoint existing = AppointRepository.SelectByID(id);

if (existing == null)

{

return new HttpNotFoundResult("Invalid Centre ID");

}

ViewData["currentMOTCentre"] = existing.MOTCentre.CentreName;

VechAppointViewModel vavm = convVAtoVAVM(existing);

vavm.MOTCentres = MOTRepository.SelectAll();

return View(vavm);

}

// Edits an appointment - handles the response back

[HttpPost]

public ActionResult Edit(VechAppointViewModel obj)

{

if (ModelState.IsValid)

{ // check valid state and time

if (isTimeValid(obj.VechAppointTime, obj.VechAppointDate.Value, obj.MOTCentresCentreId))

{

if (isRegValid(obj.VechRegNo, obj.VechAppointId))

{

VechAppoint updatedAppoint = convVAVMtoVA(obj);

AppointRepository.Update(updatedAppoint);

AppointRepository.Save();

return RedirectToAction("Index");

}

else

{

ModelState.AddModelError("Reg", "There is already a Vehicle with that Registration Number.");

obj.MOTCentres = MOTRepository.SelectAll();

return View(obj);

}

}

else

{

ModelState.AddModelError("Time", "Sorry but the Centre you have selected is not open at this time. Please refer to the MOT Centres Table and reselect a time.");

obj.MOTCentres = MOTRepository.SelectAll();

return View(obj);

}

}

else // not valid so redisplay

{

obj.MOTCentres = MOTRepository.SelectAll();

return View(obj);

}

}

// Deletes an appointment

[HttpGet, ActionName("Delete")]

public ActionResult ConfirmDelete(int id)

{

VechAppoint existing = AppointRepository.SelectByID(id);

if (existing == null)

{

return new HttpNotFoundResult("Invalid Centre ID");

}

return View(existing);

}

// Deletes an appointment - handles the response back

[HttpPost]

public ActionResult Delete(int id)

{

AppointRepository.Delete(id);

AppointRepository.Save();

return RedirectToAction("Index");

}

// Converts a VechAppointViewModel to a VechAppoint

public VechAppoint convVAVMtoVA(VechAppointViewModel vm)

{

VechAppoint va = new VechAppoint { MOTCentresCentreId = vm.MOTCentresCentreId, VechAppointId = vm.VechAppointId, VechOwner = vm.VechOwner, VechRegNo = vm.VechRegNo };

va.VechAppointTime = vm.VechAppointDate.Value.Date + vm.VechAppointTime.TimeOfDay;

return va;

}

// Converts a VechAppoint to a VechAppointViewModel

public VechAppointViewModel convVAtoVAVM(VechAppoint va)

{

VechAppointViewModel vm = new VechAppointViewModel { MOTCentresCentreId = va.MOTCentresCentreId, VechAppointId = va.VechAppointId, VechOwner = va.VechOwner, VechRegNo = Regex.Replace(va.VechRegNo, @"\s+", "") };

vm.VechAppointTime = DateTime.Parse(va.VechAppointTime.ToString("HH:mm"));

vm.VechAppointDate = va.VechAppointTime.Date;

return vm;

}

/\* Checks that the appointment time is valid

\* i.e. 3pm on the 23/07 - what day is it and does 3pm fall within opening hours

\*

\* for each time in list

\* if the week day is correct and the time is within the range then all's good

\* otherwise fail

\*/

public bool isTimeValid(DateTime time, DateTime date, int centreId)

{

if (MOTRepository.GetTimes(centreId).Any())

{

foreach (var ct in MOTRepository.GetTimes(centreId))

{

if (ct.DayOfTheWeek == (int)date.DayOfWeek)

{

if (time.TimeOfDay > ct.OpeningTime && time.TimeOfDay < ct.ClosingTime)

{

return true;

}

}

}

return false;

}

return false;

}

public bool isRegValid(string regNo, int id = -1)

{

if (AppointRepository.SelectAllRegs(id).Any())

{

foreach (var reg in AppointRepository.SelectAllRegs(id))

{

if (String.Compare(regNo, Regex.Replace(reg, @"\s+", "")) == 0)

{

return false;

}

}

}

return true;

}

}

}

### Design Rational

The Index method takes an optional parameter which is used for displaying appointments for specific centres. If left as is all appointments will be displayed. If a ID is given, just the appointments associated with that centre will be displayed.

Like the MotCentreController any functions that take an ID will return a 404 error if that ID is invalid.

Also like the MotCentreController the VechAppointController uses two classes to separate validating and displaying information. It also implements conversion functions to change between these structures. This was done to allow greater amounts of validation control over each piece of data entered into the system. The flow of data is identical to the one used in the MotCentreController just with the classes changed.

When creating and editing a centre, a list of the current MOT Centre names is passed into the VechAppointViewModel. This is used to create a drop down list that allows the user to easily select the centre they wish to book with. The Edit method is set up so that the current value of MOT Centre will already be selected by the drop down to prevent data being corrupted.

On return the data is validated with the annotations found in the CS file then the appointment time and Vehicle Registration is check. The time must fall within the Centre’s opening hours and the Registration must not already exist in the database. The Registration check excludes the current value of the vehicle registration otherwise an error would occur. Should any of these checks fail appropriate error messages are returned to the user.

Helper methods such as conversion functions are used alongside the validation functions for the vehicle registration number and appointment time. Again these are useful as they reduce the code in each of the HTTP functions and help keep unrelated functions separate.

## Mot Centre Opening Hours

To store the opening hours of a centre, a string where a user could type how the hours functioned could be sufficient but I felt it was far to variable and could end up having to support a wide range of display styles along with other problems. I reasoned that a second table and class should be created to handle this which is called the CentreTime class.

namespace Assignment2MOT

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

public partial class CentreTime

{

public int CentreTimeId { get; set; }

public int MOTCentresCentreId { get; set; }

public int DayOfTheWeek { get; set; }

public TimeSpan OpeningTime { get; set; }

public TimeSpan ClosingTime { get; set; }

public virtual MOTCentre MOTCentre { get; set; }

}

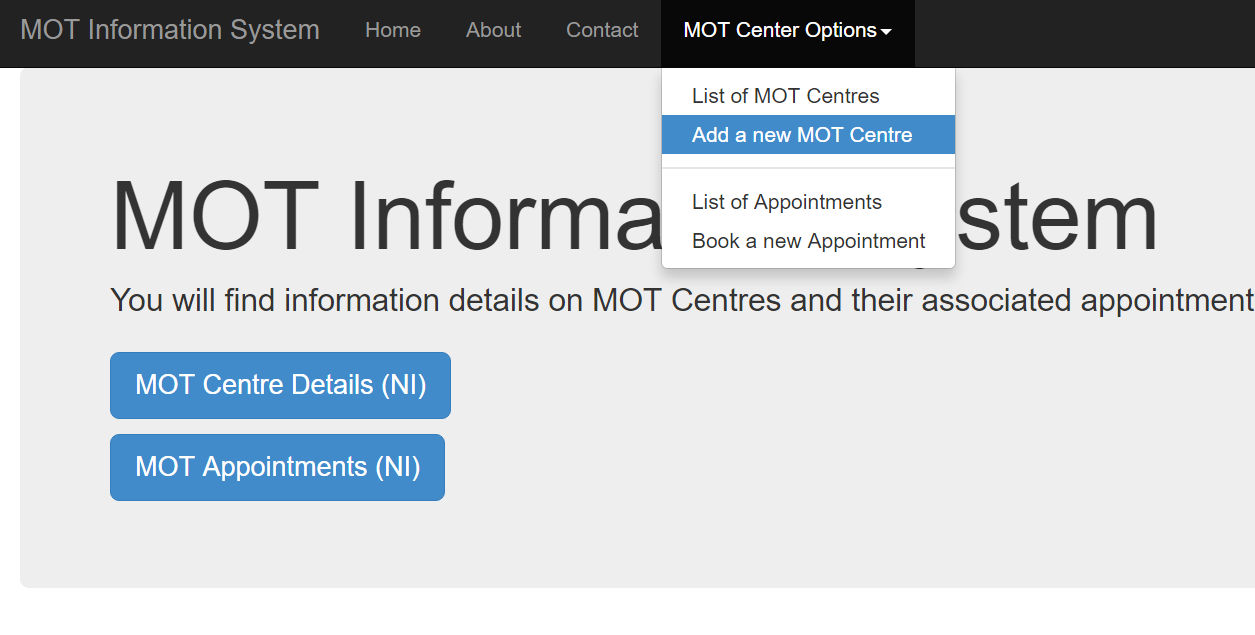
}

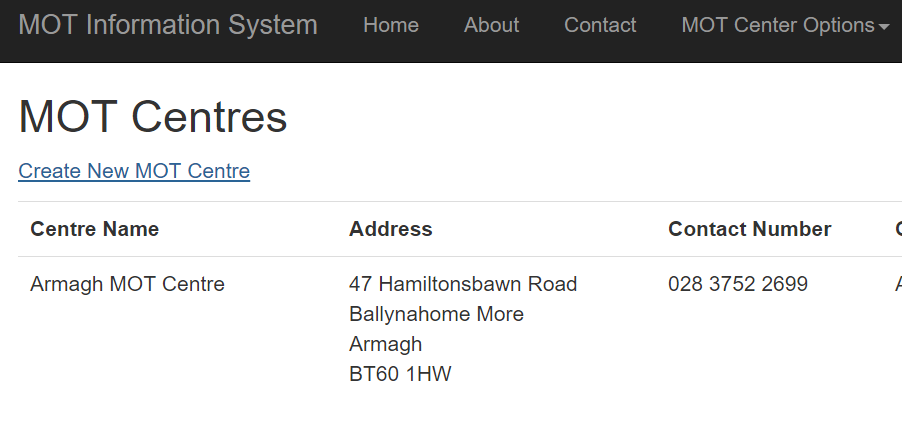
The class is simple and the standard functionality is each MOT Centre will have between 1 and 7 records for each day of operation. To save on storage space, the records will be created and deleted as needed (i.e. if the centre is open for 3 days then it will have 3 entries).

Its operation is mostly used alongside the MOTCentre class as they are tied quite closely together. Validation is not required in the class as it is handled completely in the Controller.

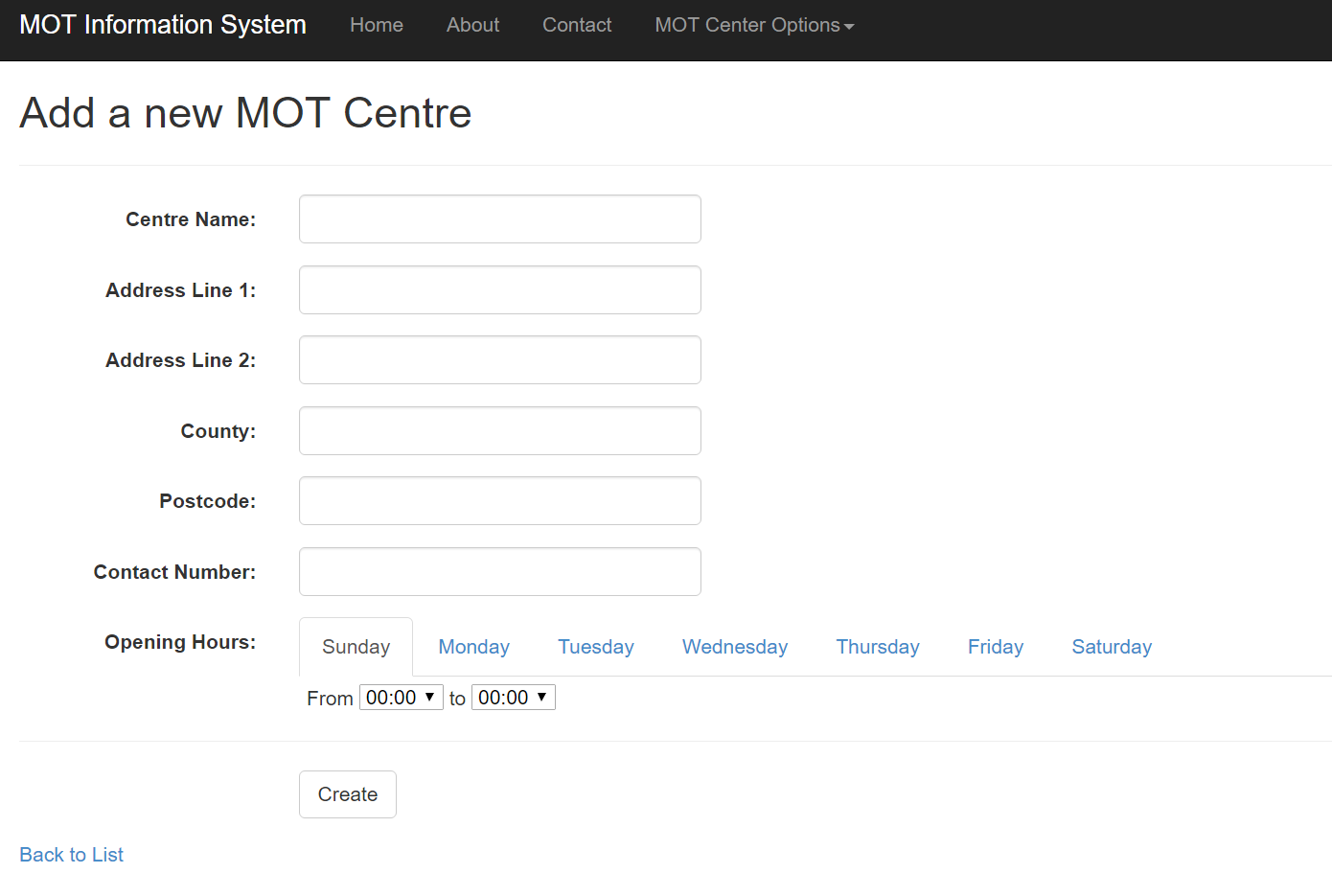
## MotCentre Demo

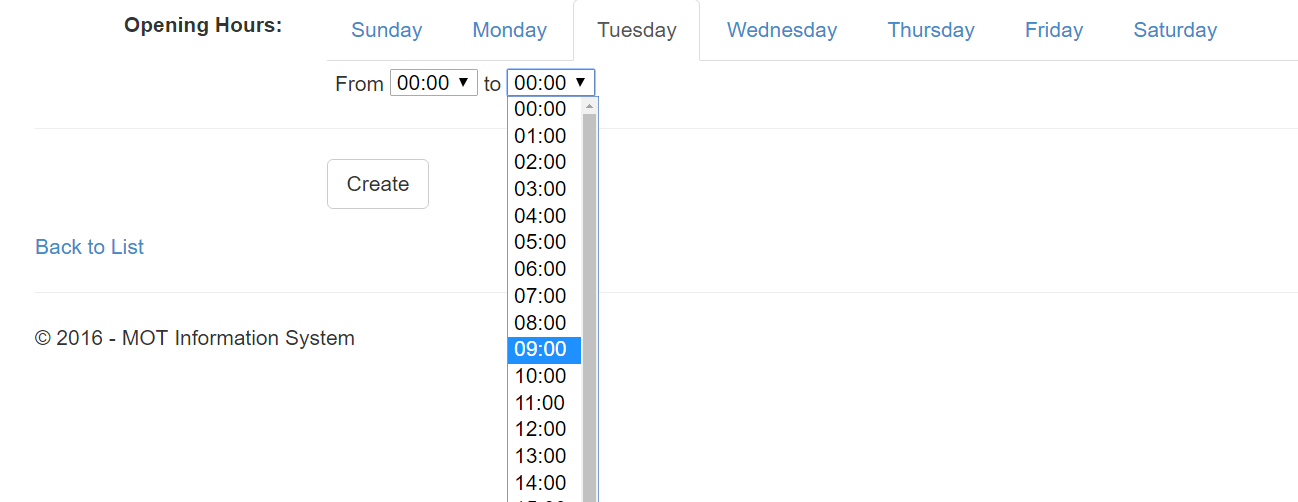
### Create a MOTCentre



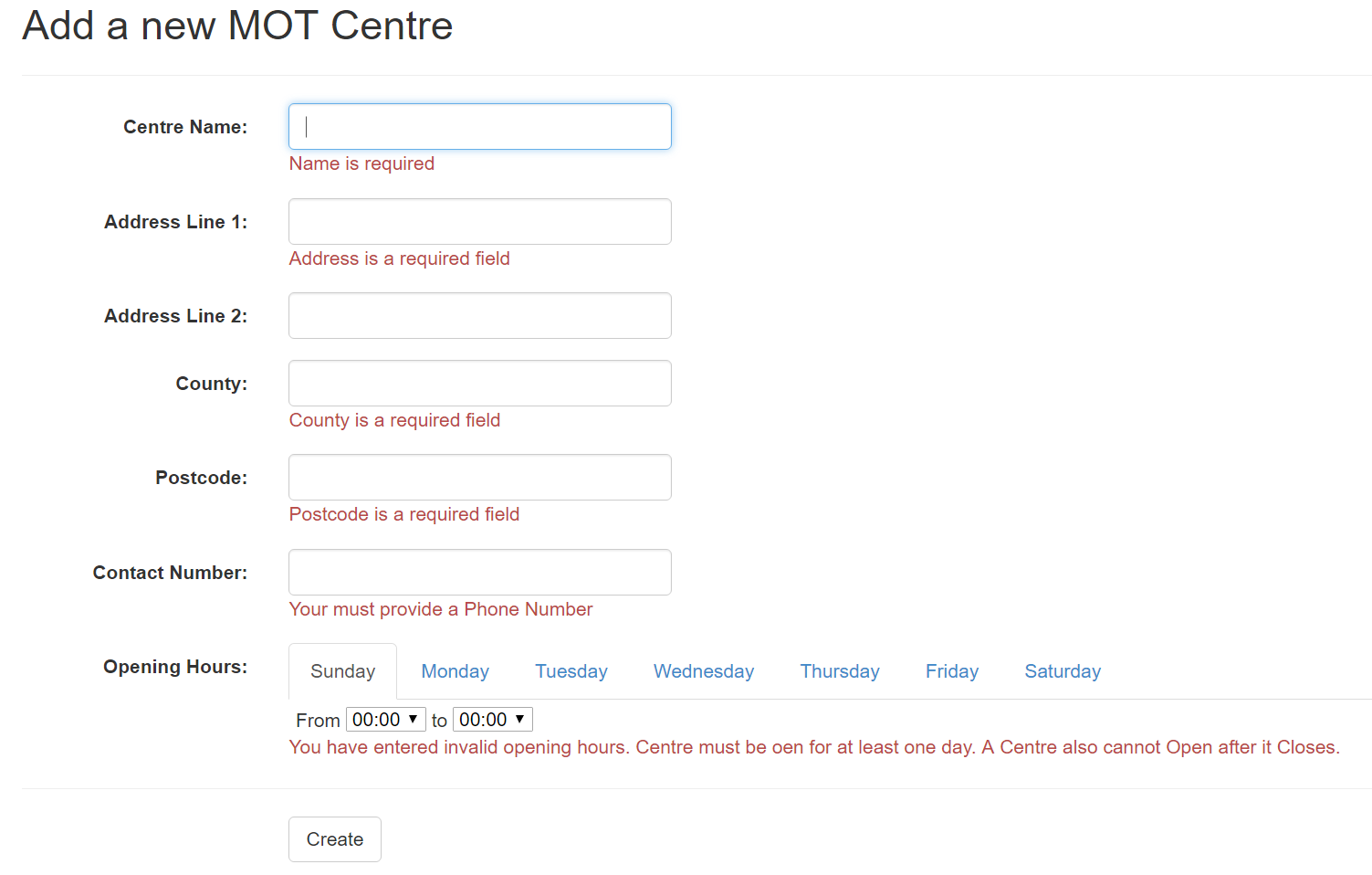


You can create a centre through either the drop down bar across every page or from the “Create a New MT Centre” link in the MOT Centres table.

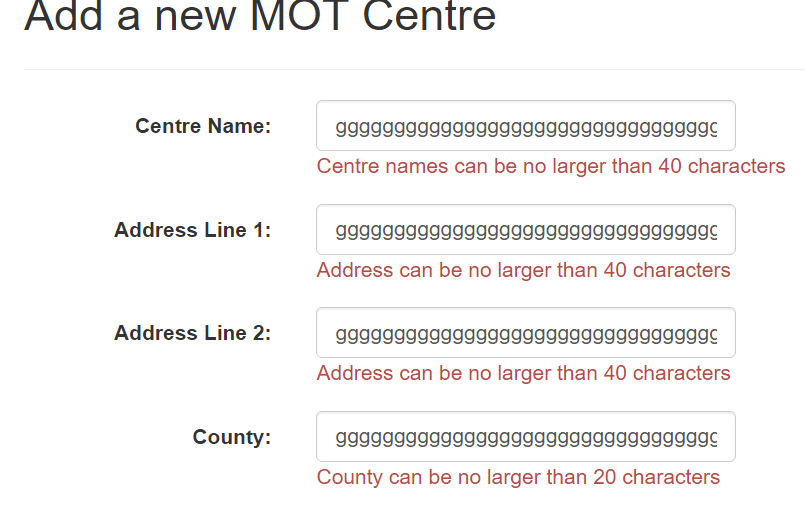




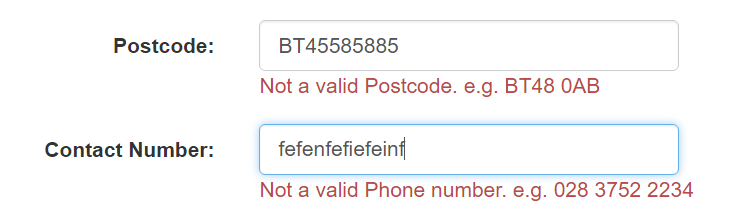
Each centre has a number of fields though not all required to be filled. The second address line is optional and the centre can be open for any number of days.



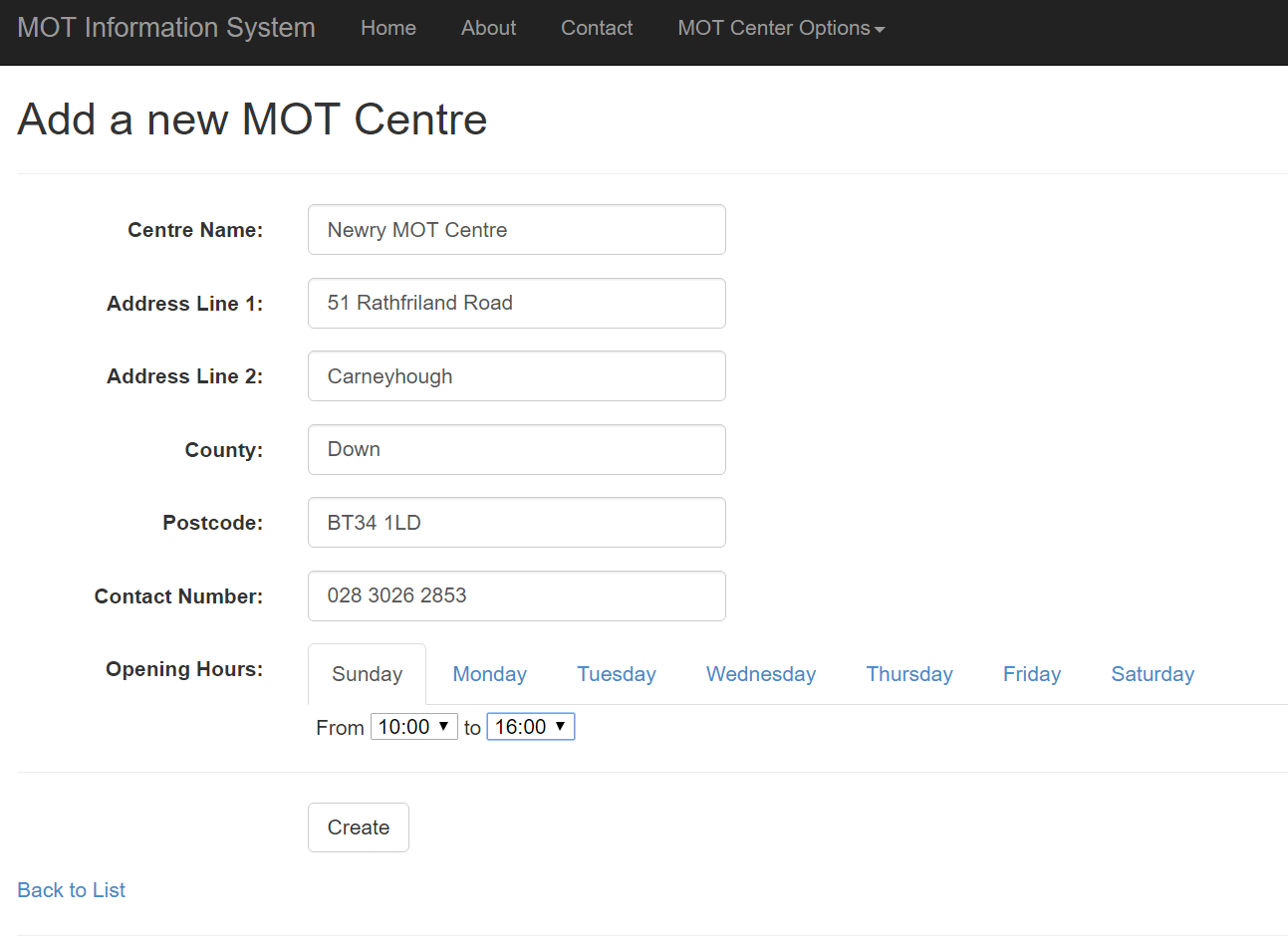
As shown most fields are required for the successful creation of a centre record. One possible improvement here would be to change the County text box to a drop down allowing the user to select their county instead. This was not implimented due to lack of time. Another improvement could have been the implimentation of a Reset or Clear button which cleared all data currently entered into the form.

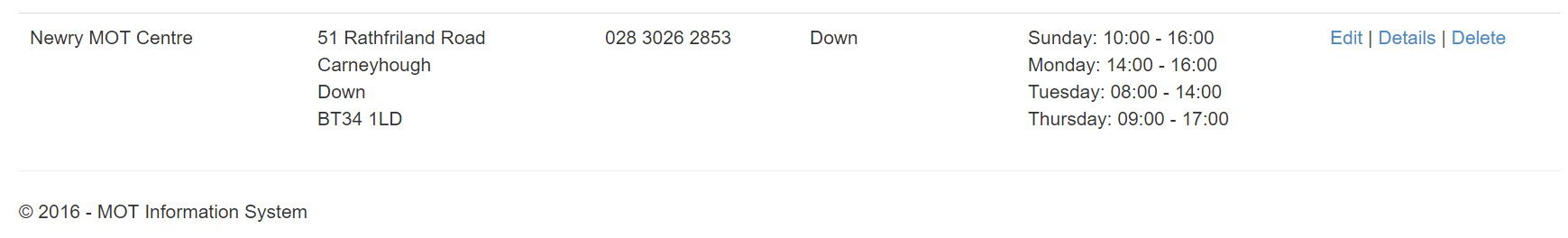


Character limits are applied to all fields to prevent unnessicarly large data entries.



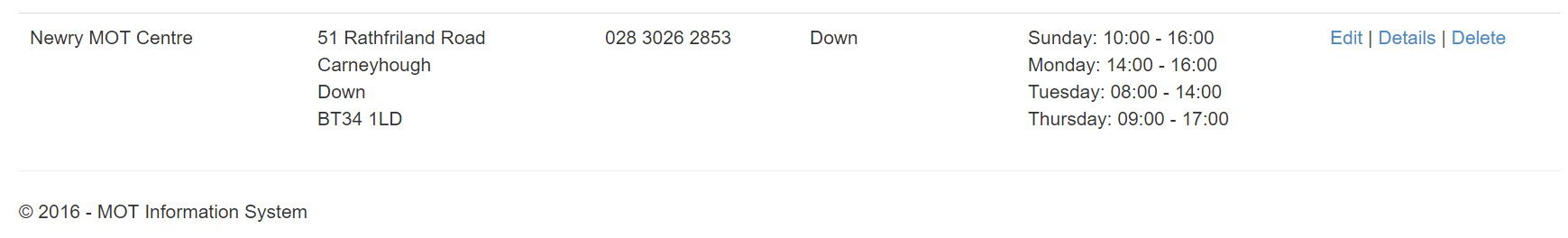
The Postcode and Contact Number use regular expressions to limit input into vaild entries. Postcode only accepts valid Northern Ireland postcodes i.e. BT1 to BT94. Contact Number will accept a 11 digit phone number with or without spaces as validation code removes any invalid characters such as spaces.



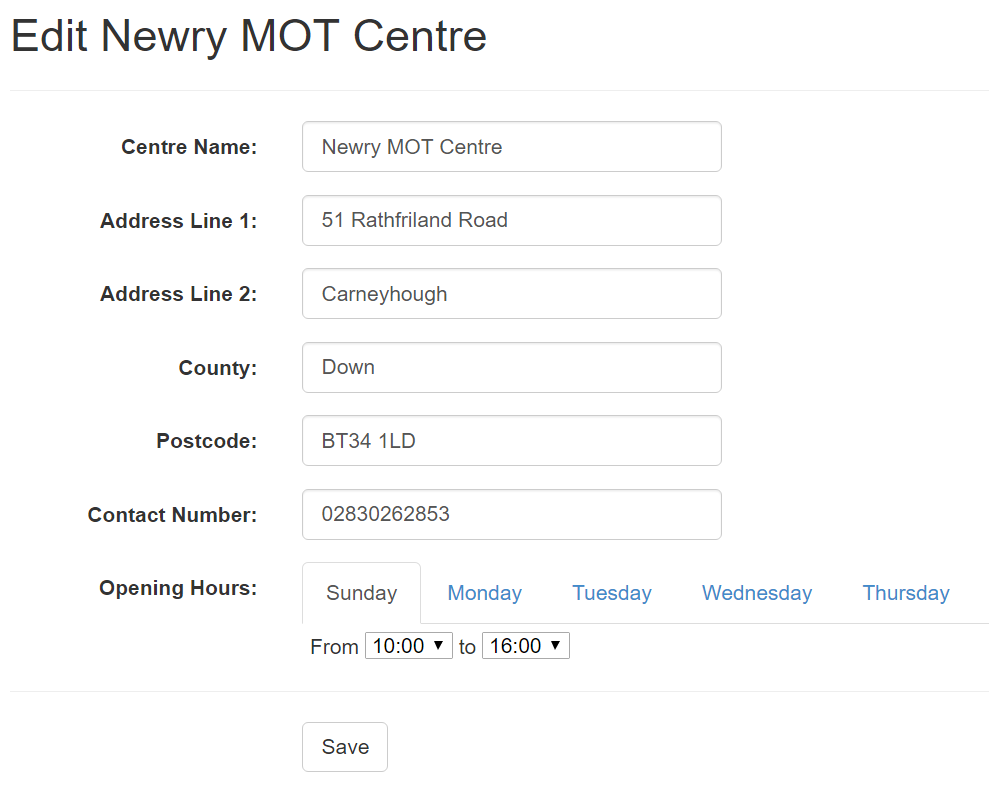


Once a valid entry is made it is added to the table.

### Edit a MOTCentre



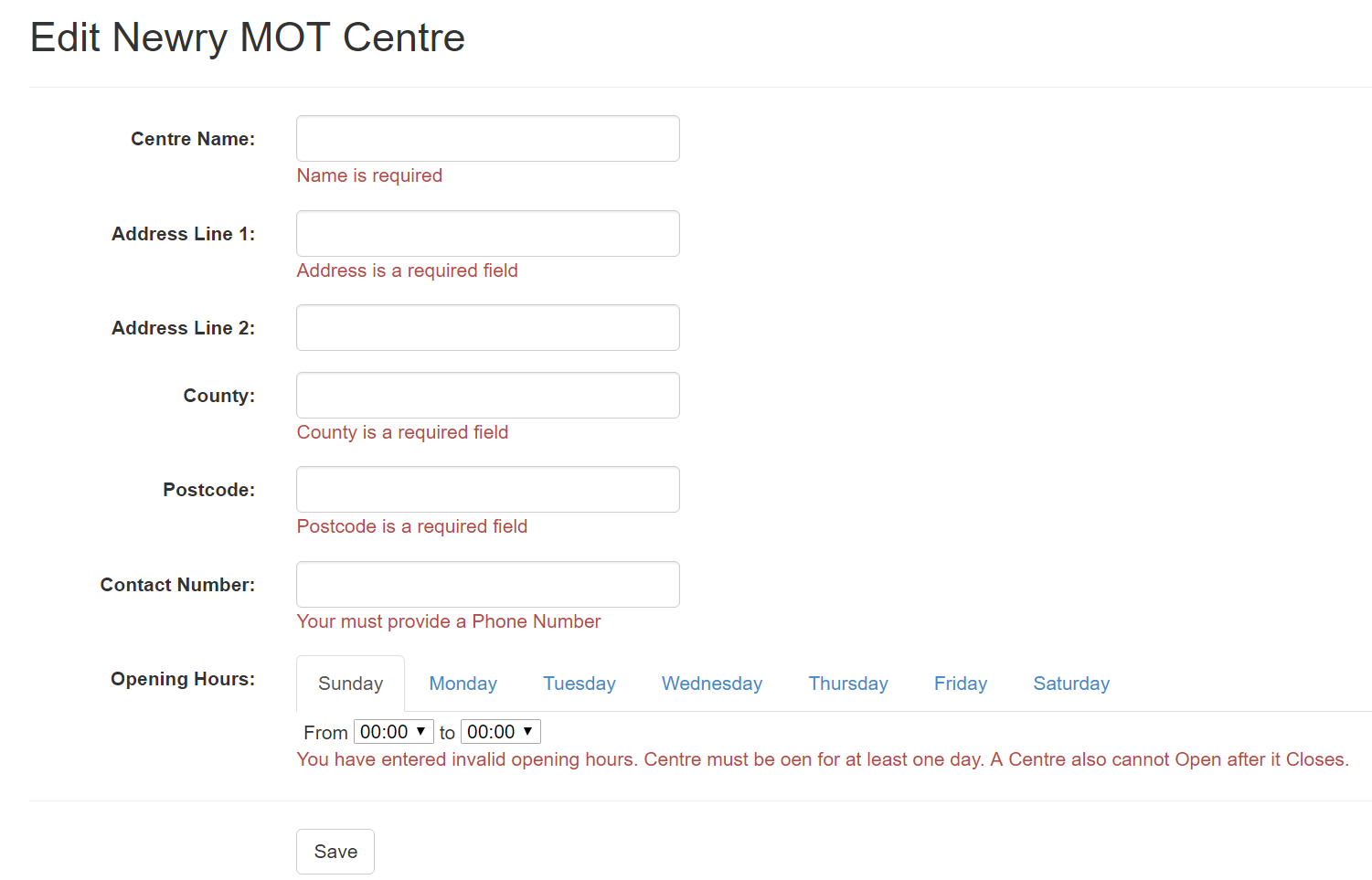
To edit an MOT Centre the user simply clicks the Edit button to the right of a record.

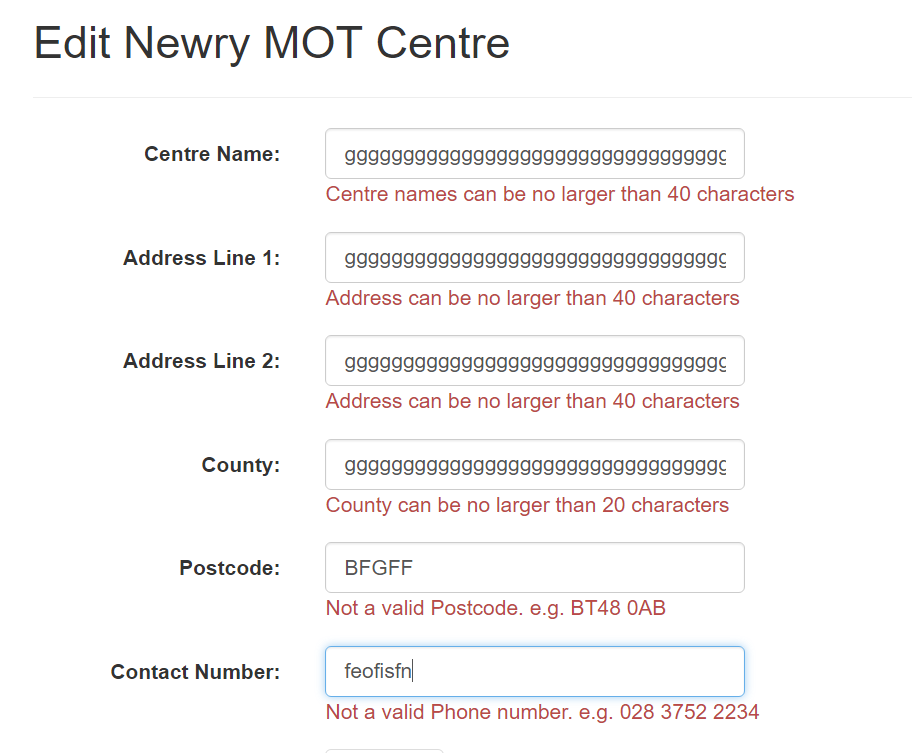


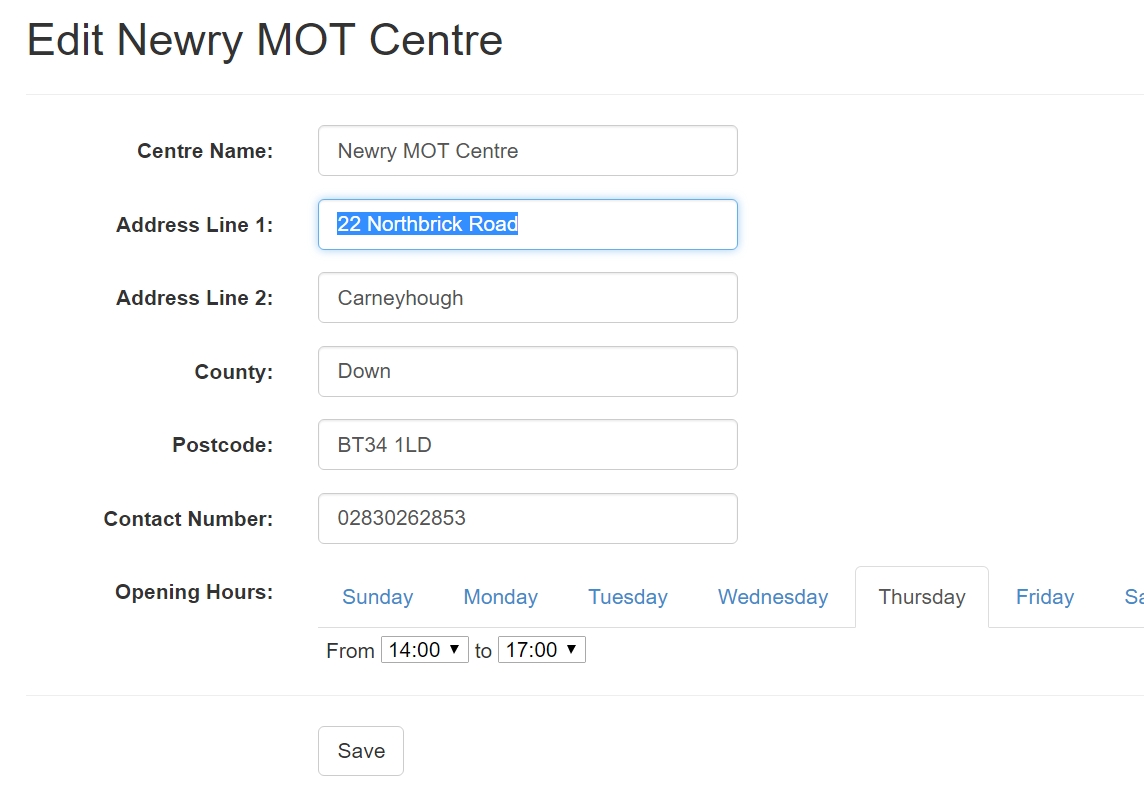
All fields will be populated with the current values and have the same validation functionality as the Create Centre View.

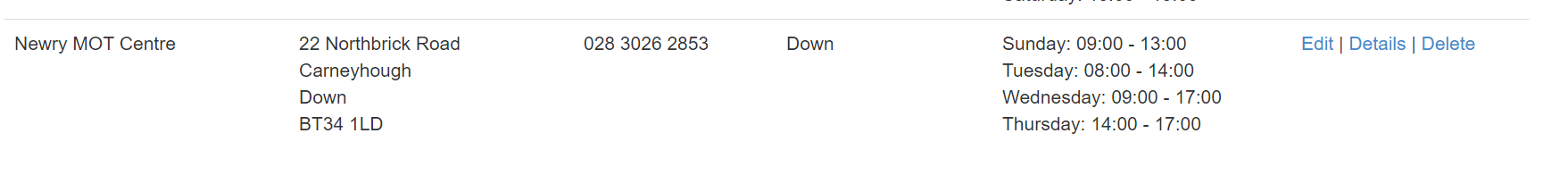
To personalise the edit page, the current centre name is displayed as part of the view heading.

Like the Create page, the Edit page would benefit from having a drop down selector in place of the County text box. The Edit page could have also implemented a reset button that clears all fields and resets all times to 00:00.



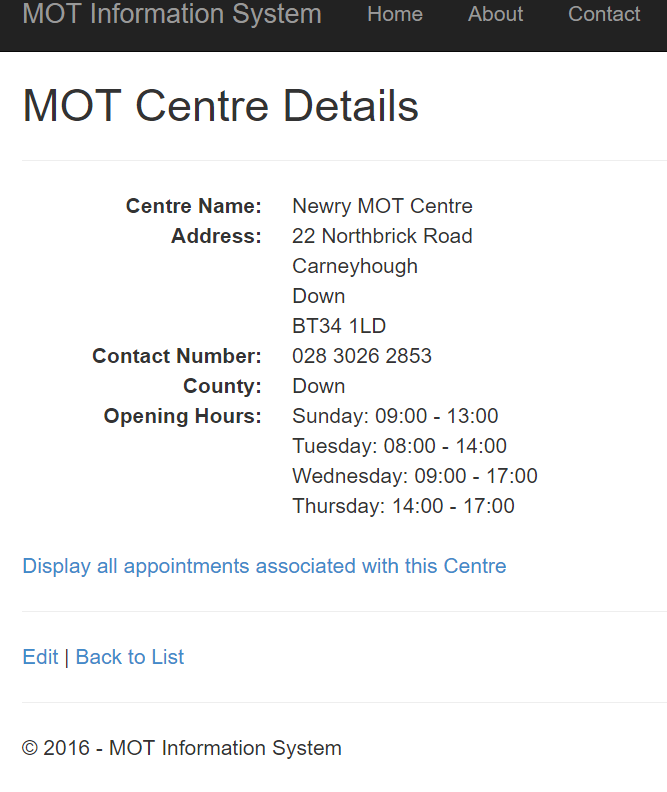






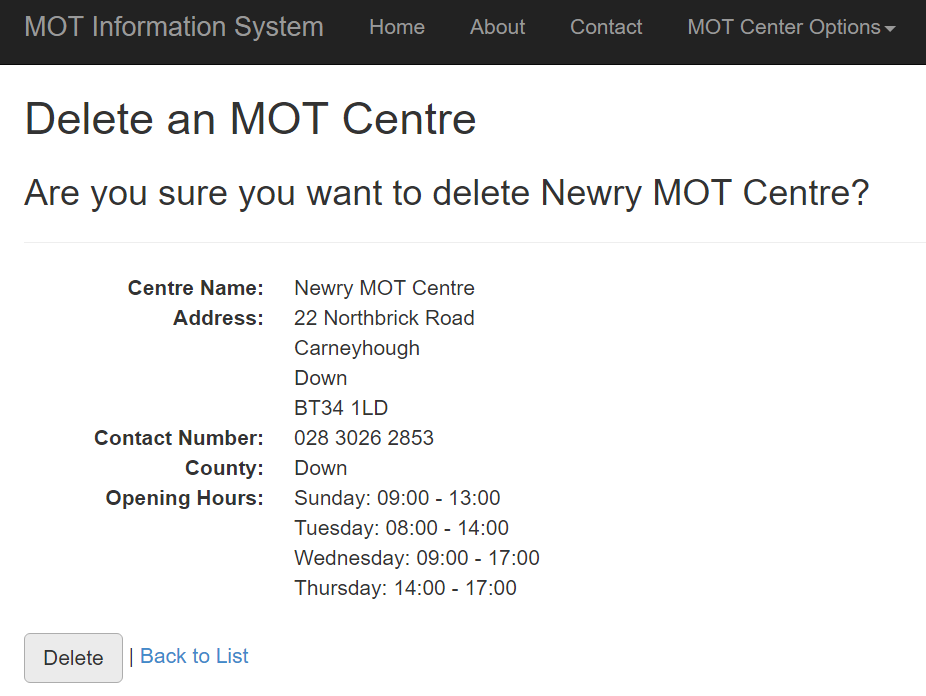
Once the data returned is valid it is saved and the database is updated. The user is then brought back to the Centre listing page.

### Details of a MOTCentre

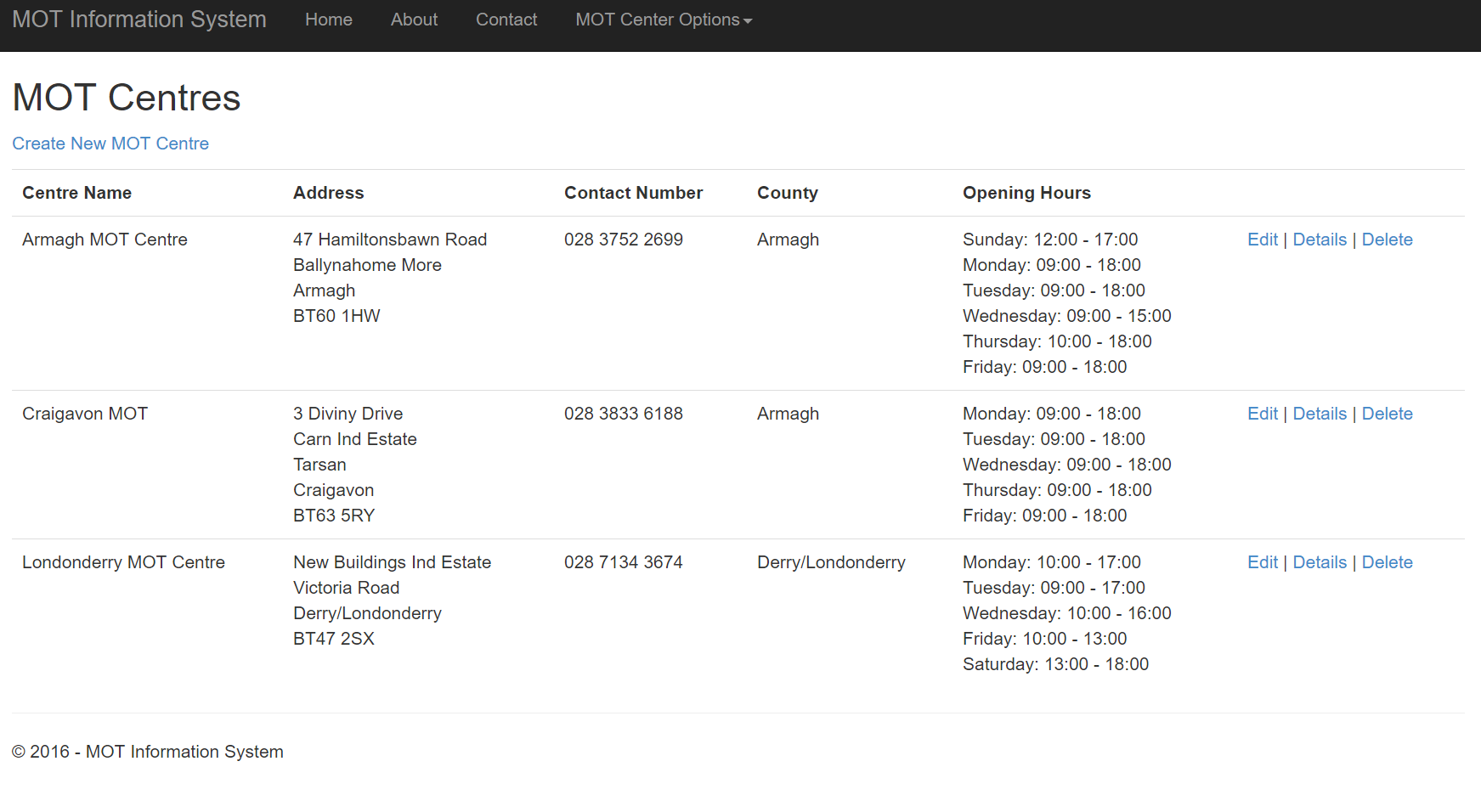


The Details page can be accessed using the Details link to the right of the record. It displays the details of the given centre and allows the option to view all appointments associated with it.

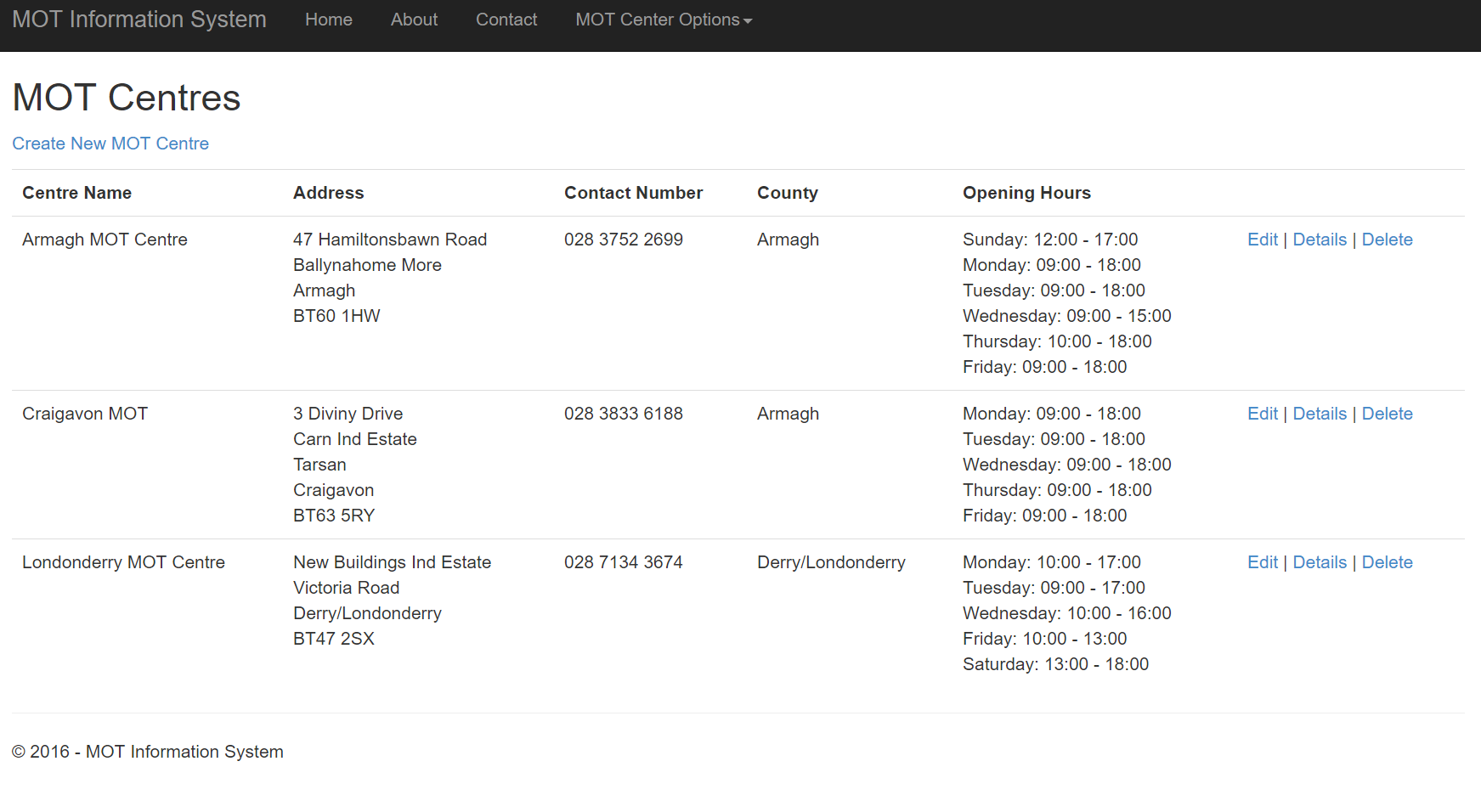
### Deleting a MOTCentre



The Delete page can be accessed using the Details link to the right of the record. Similar to the Edit page it’s headings change depending on the centre selected. Clicking the delete button will permanently delete the centre as shown below.



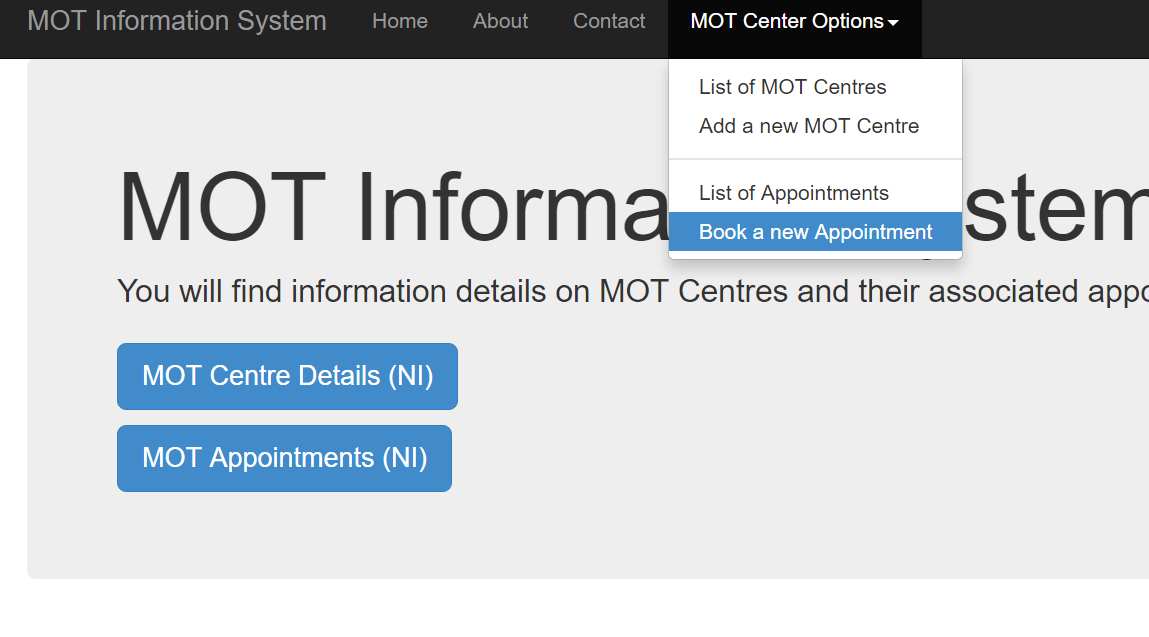
### List of MOTCentres

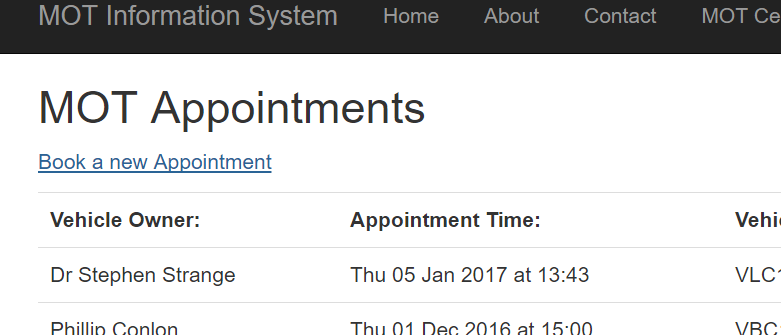


A list of all centres is returned though clicking the button on the homepage or choosing the List option from the navbar’s drop down menu. In this view the Address is neatly formatted along with the Contact number and opening hours. This is to make it easier for the user to read the table.

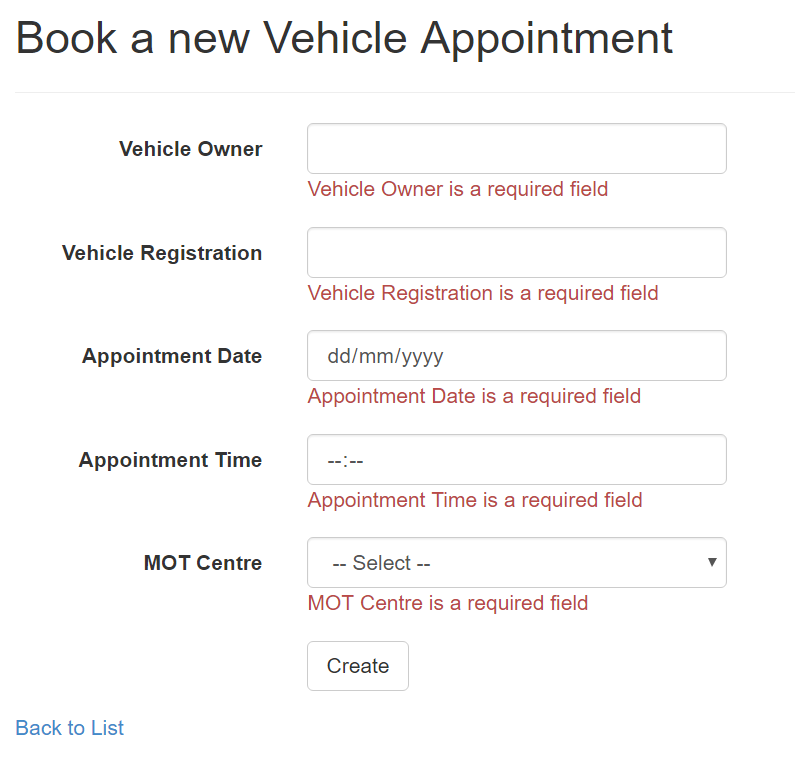
## Appointment Demo

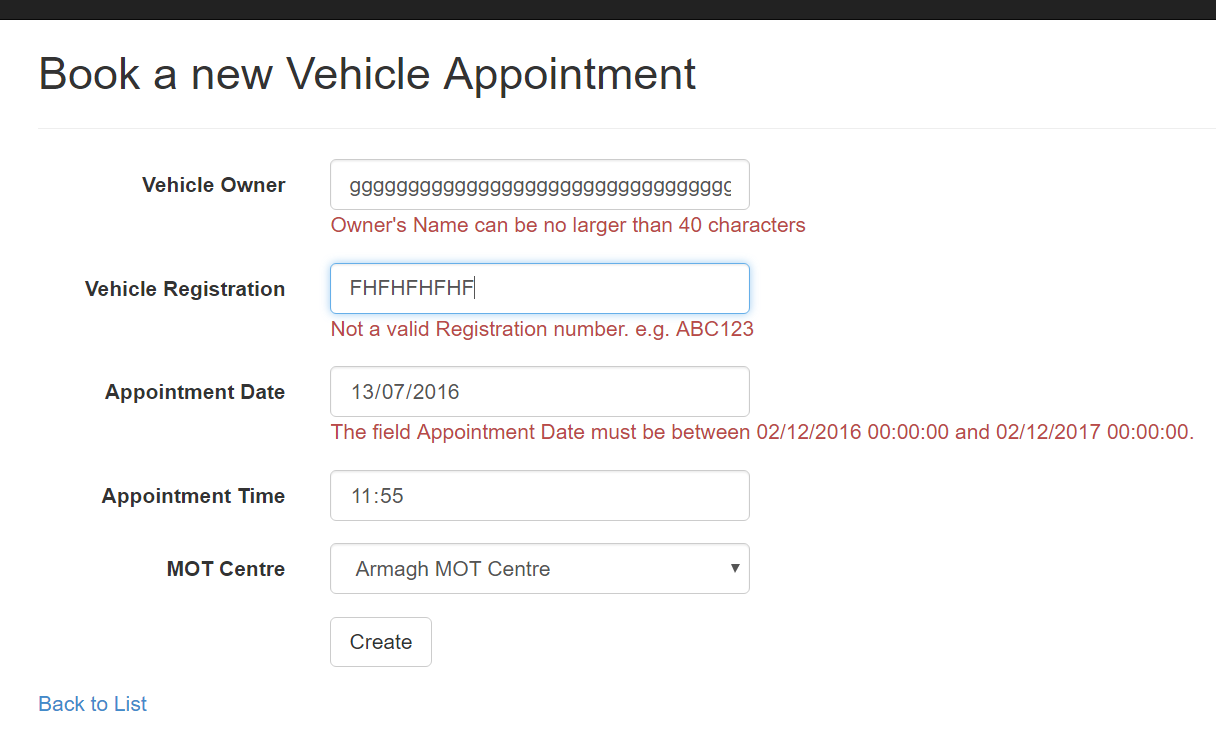
### Creating an Appointment

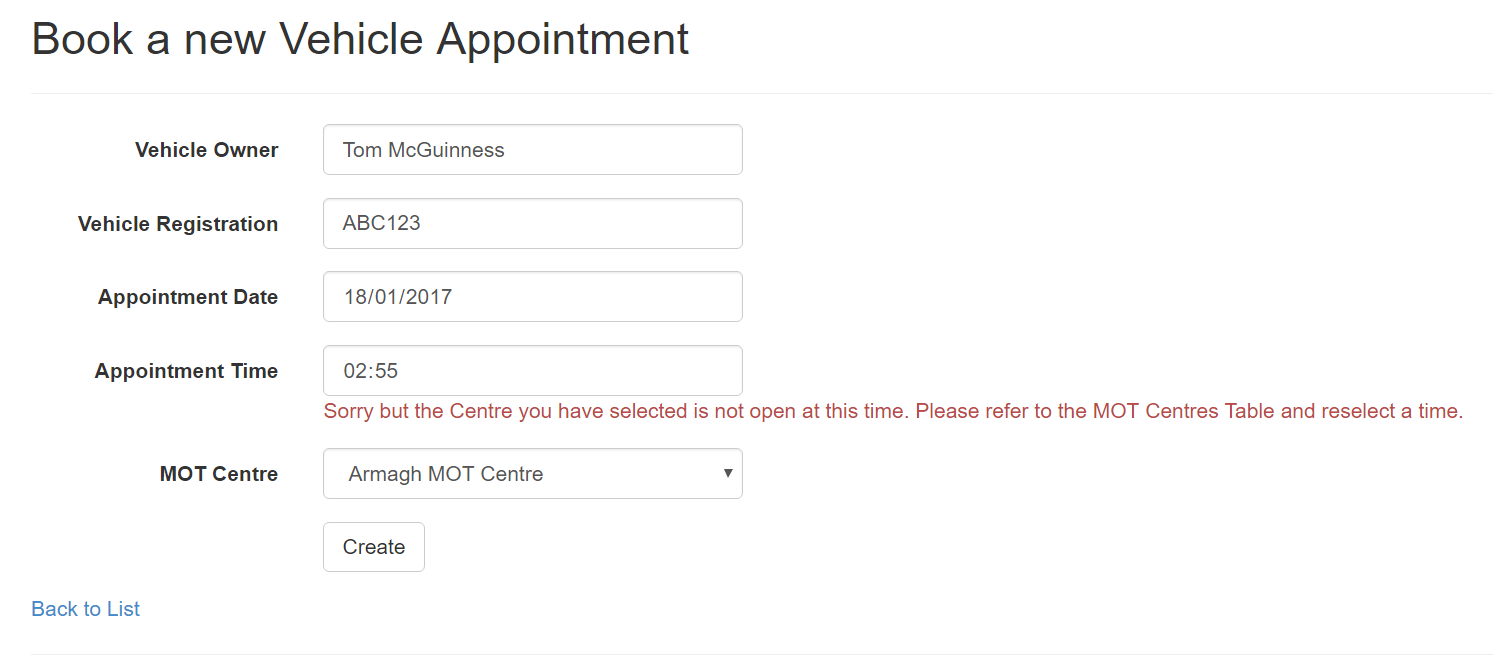


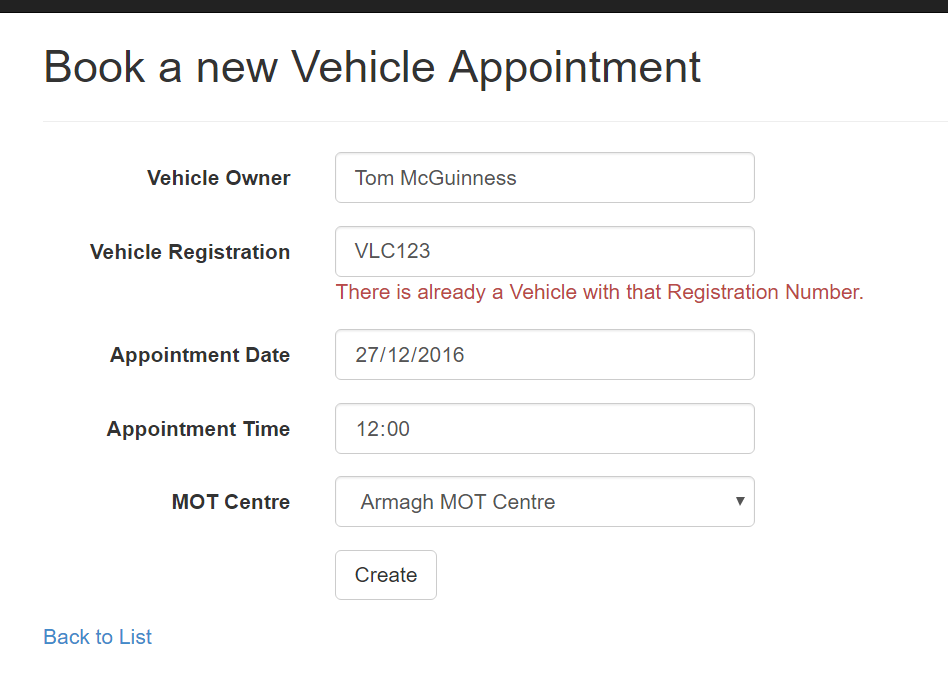


Appointments can be booked through the option in the MOT Centre Options drop down or by using the “Book a new Appointment” link on the Appointment table.



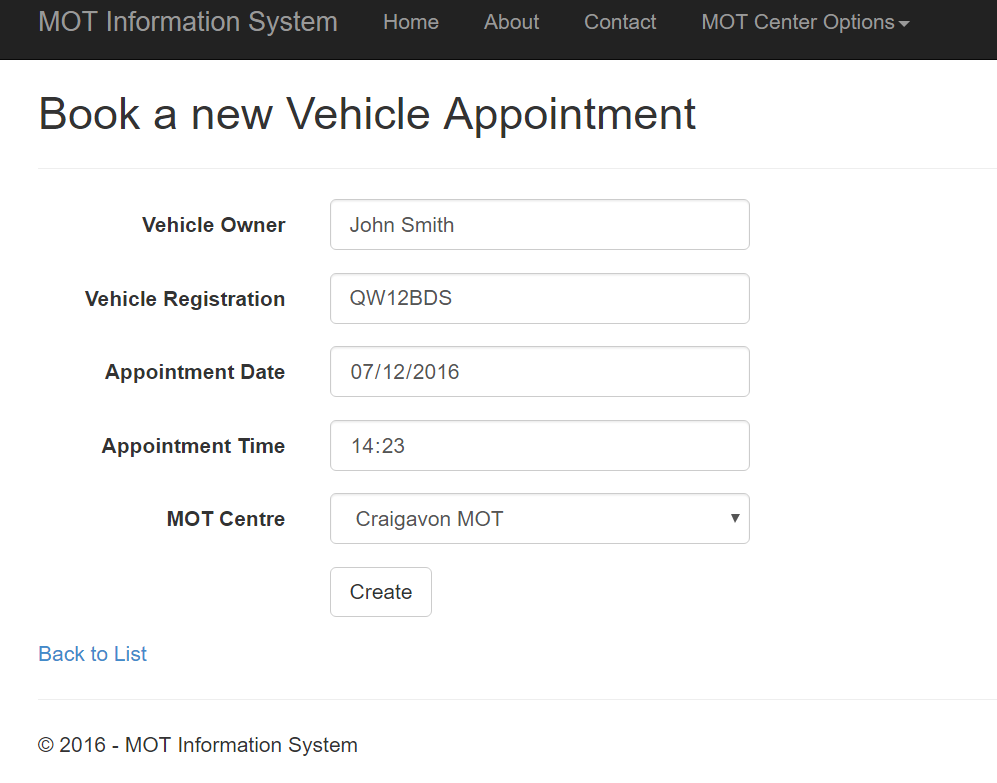


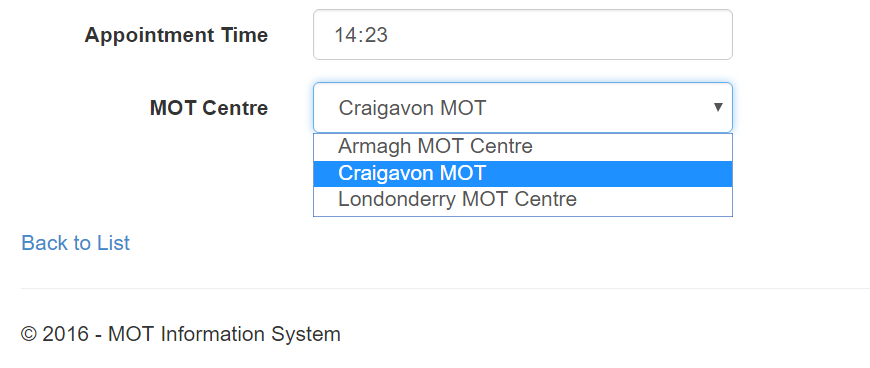




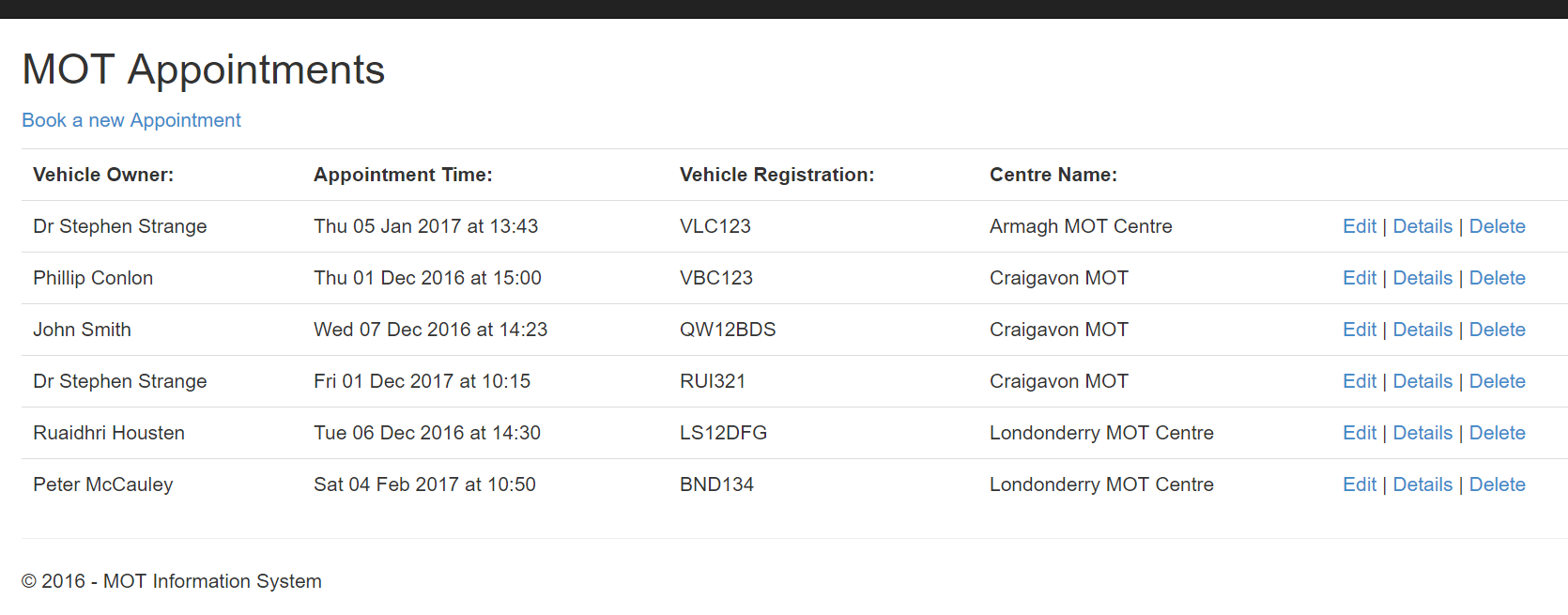
All fields in the Vehicle Appointment entries are required when booking an appointment. The Vehicle owner field is limited to 40 characters to prevent invalid names. The Registration field is validated using a regular expression that will only accept licence plate numbers that are valid according to the DVLA. Appointments cannot be booked for the past and so only dates between today and one year from now are accepted.

A check is performed to ensure the appointment time is a time when the MOT centre selected will be open. If not the user will be informed of this. The registration number is also checked to ensure no duplicate cars are entered to the database.

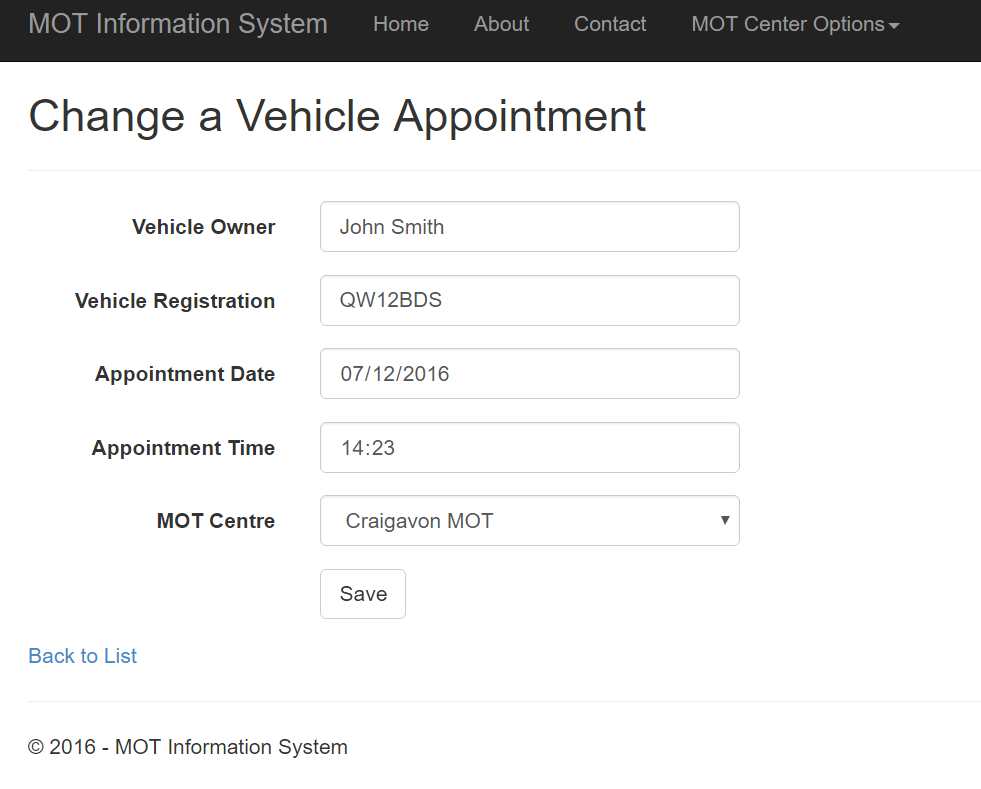


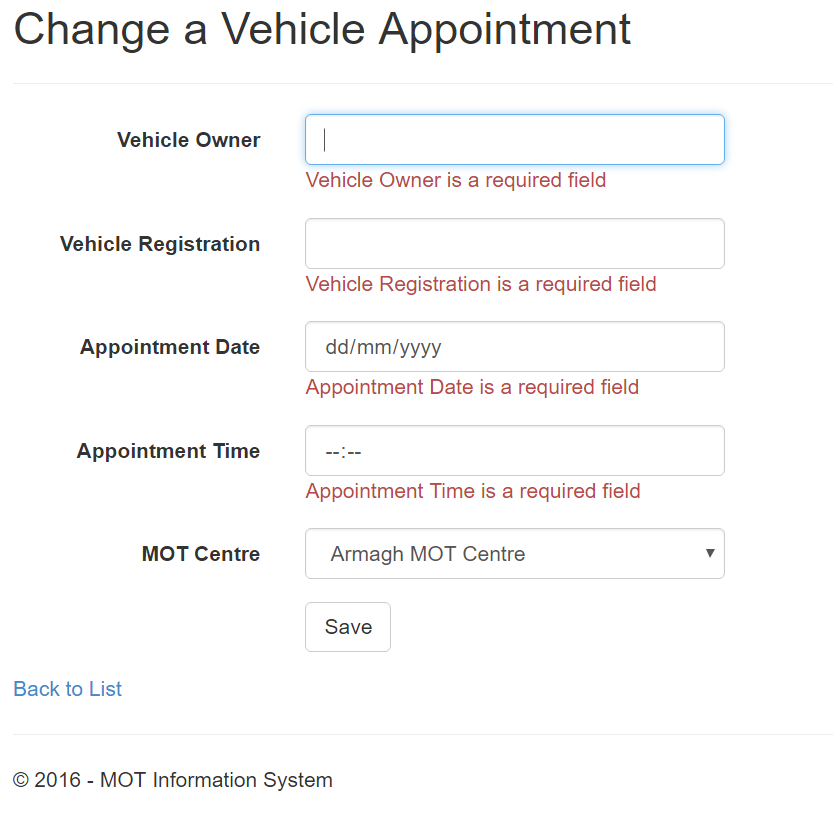


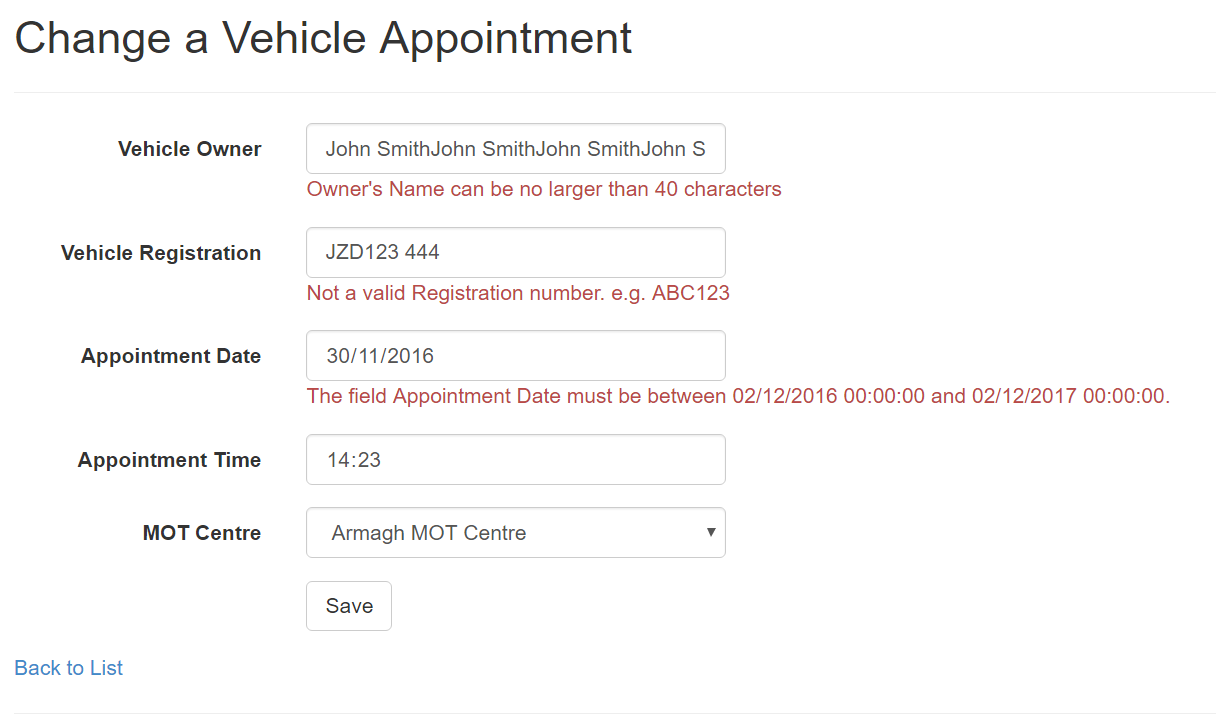
The MOT Centre is a drop down list containing all the current registered centres. Once the data that’s entered is valid a new appointment can be created. Show below is the new appointment.

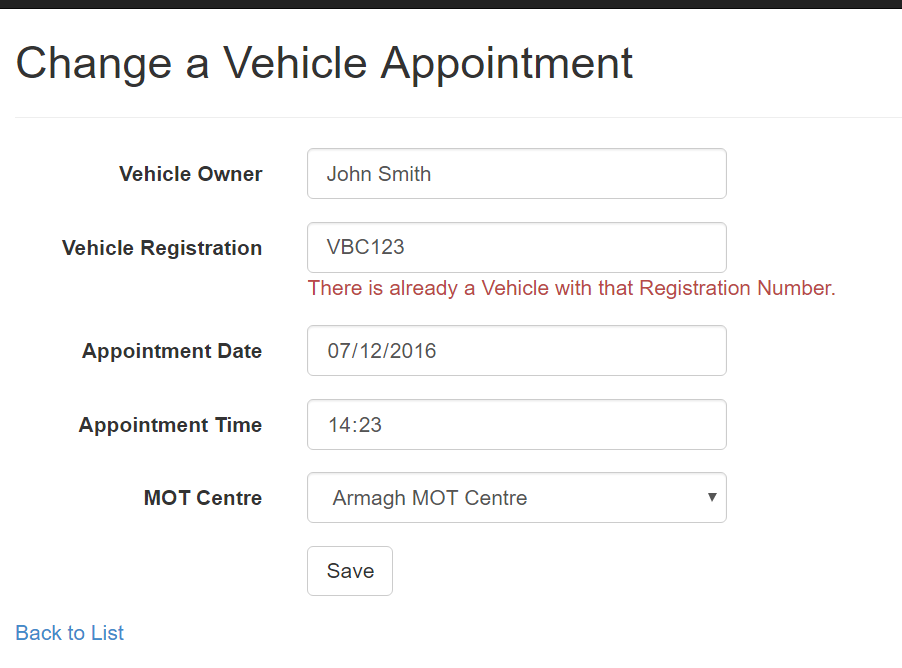


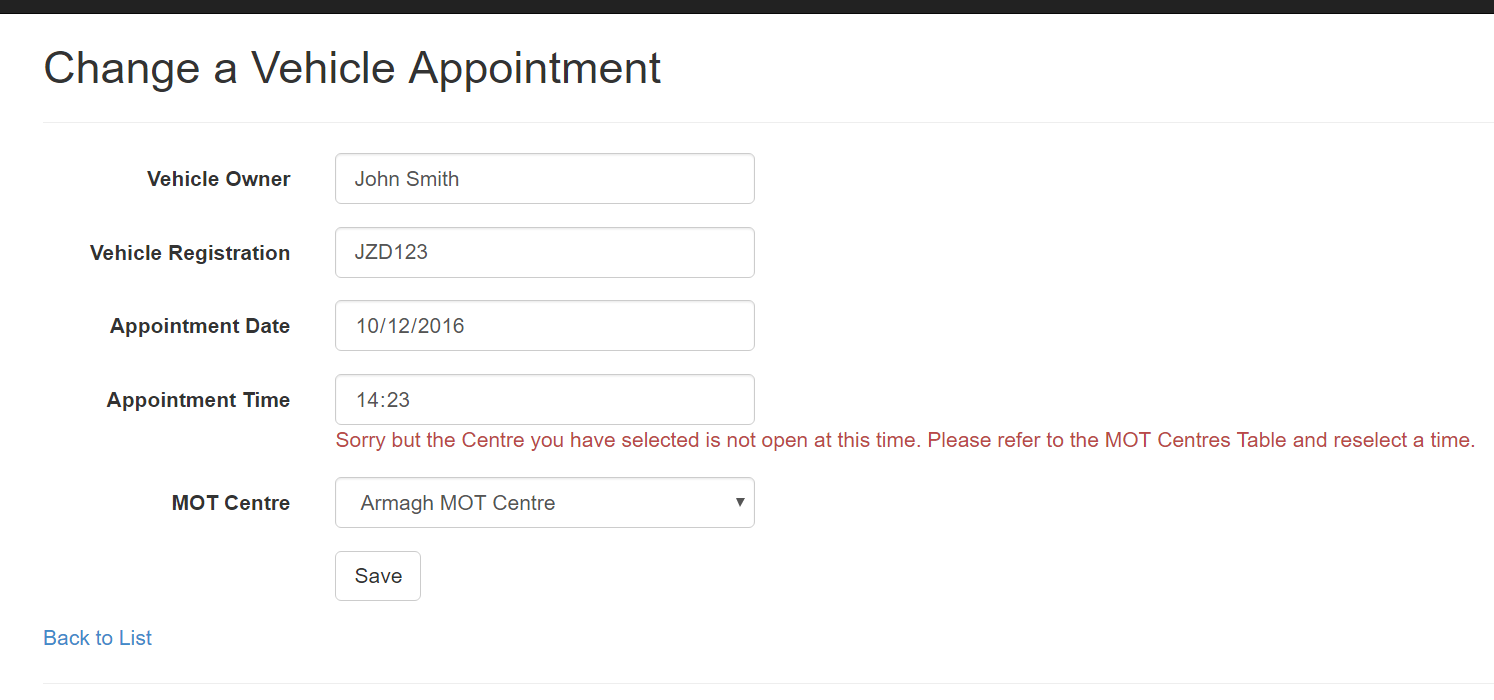
### Edit an Appointment

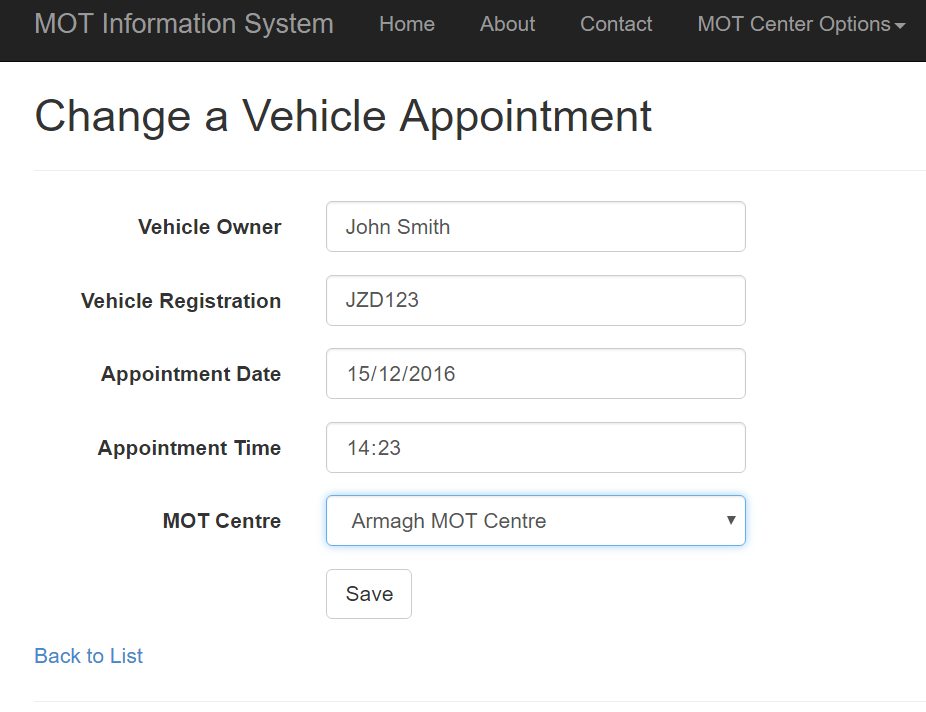




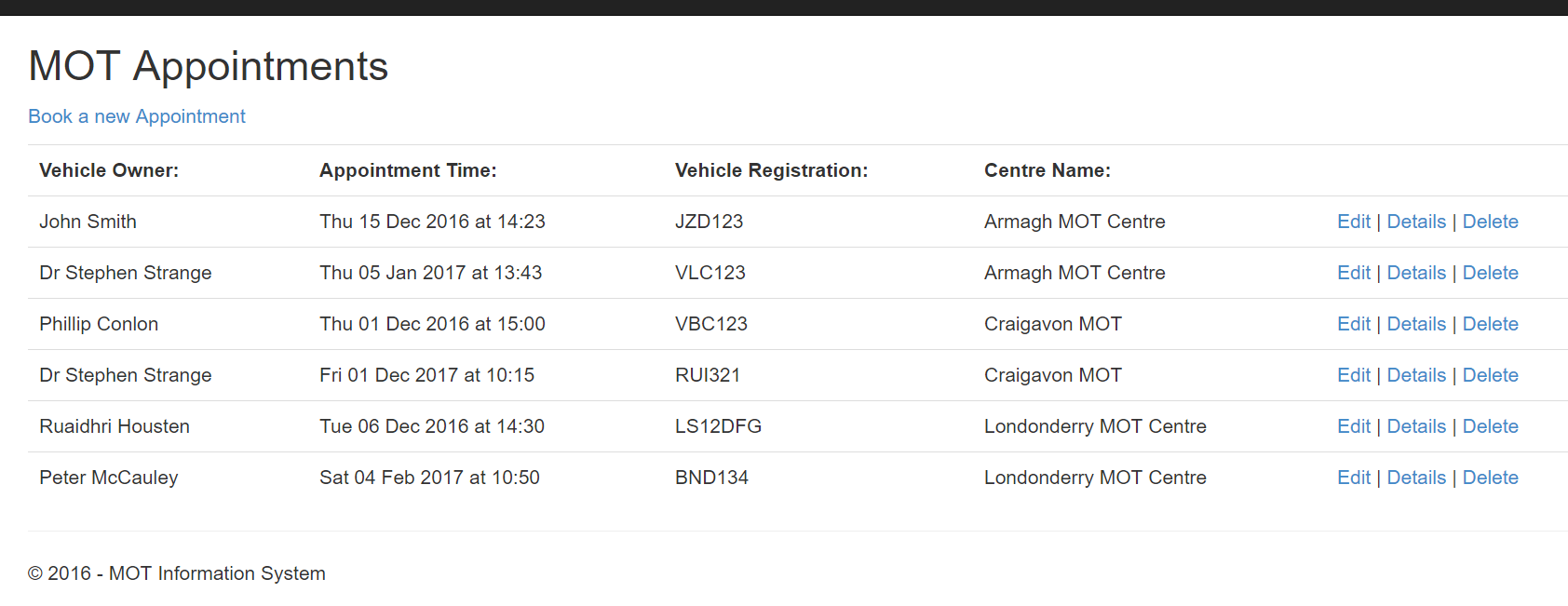




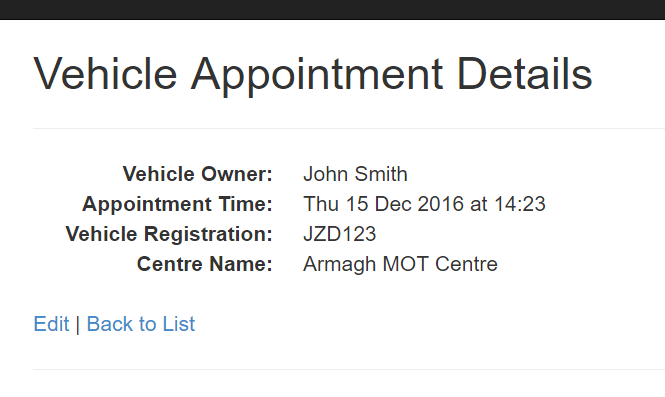




To change an appointment, the user selects the Edit link to the right of the record. The validation for the edit page is the same as that of the Create page.

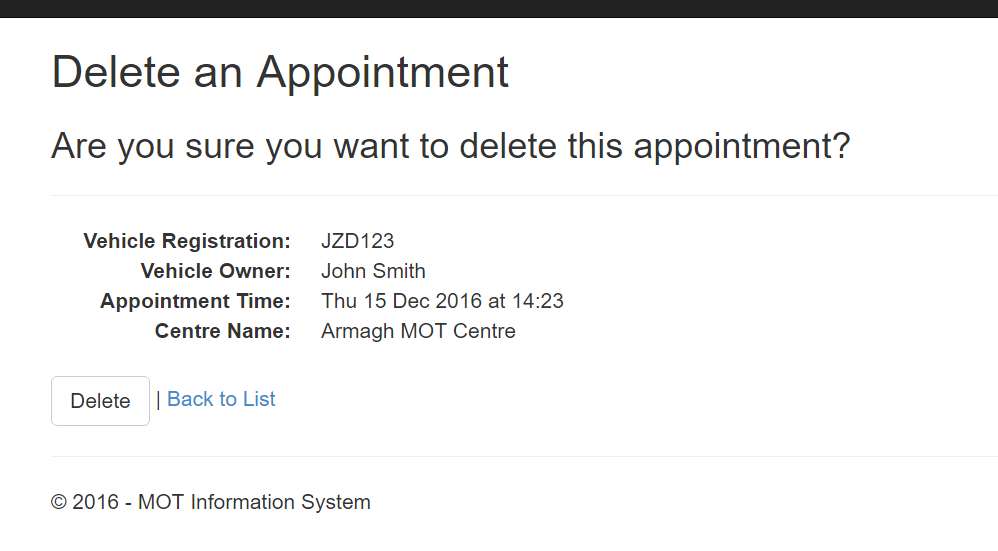


### Detailing an Appointment

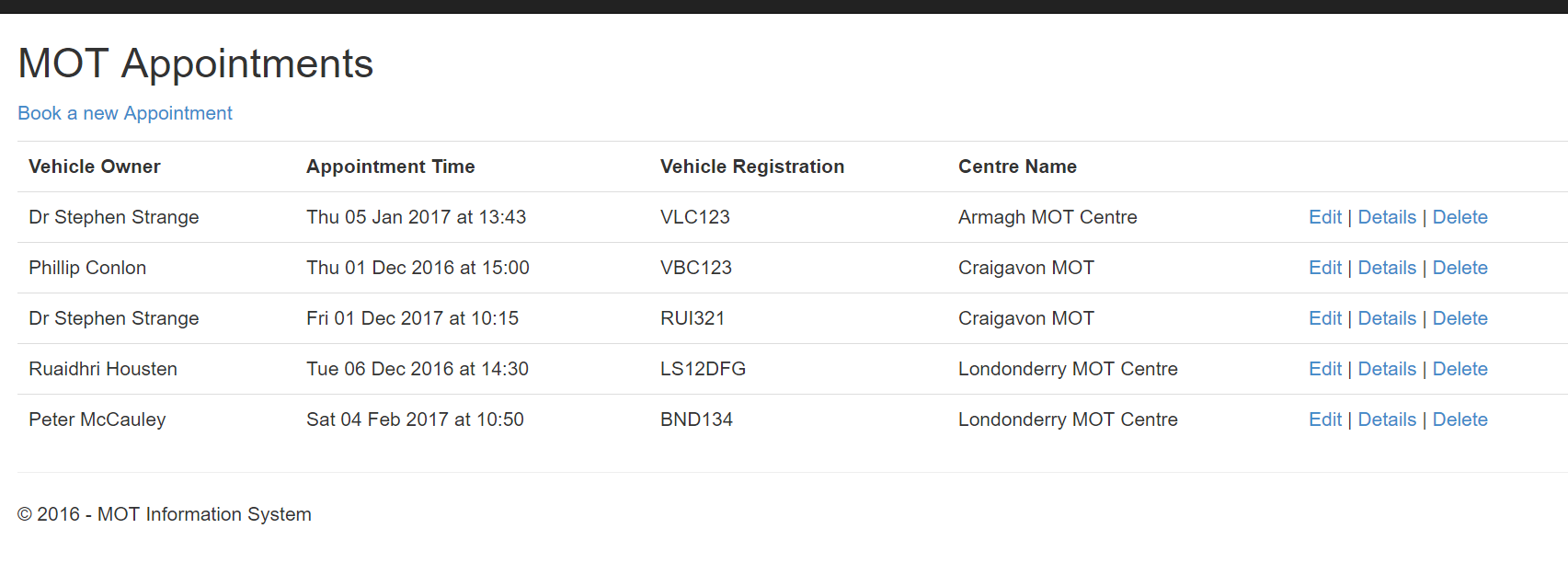


To display the details on an appointment the user can click the Details link to the right of the record. All dates use the date format from the Appointment.cs file. This gives a detailed description for the date of the appointment.

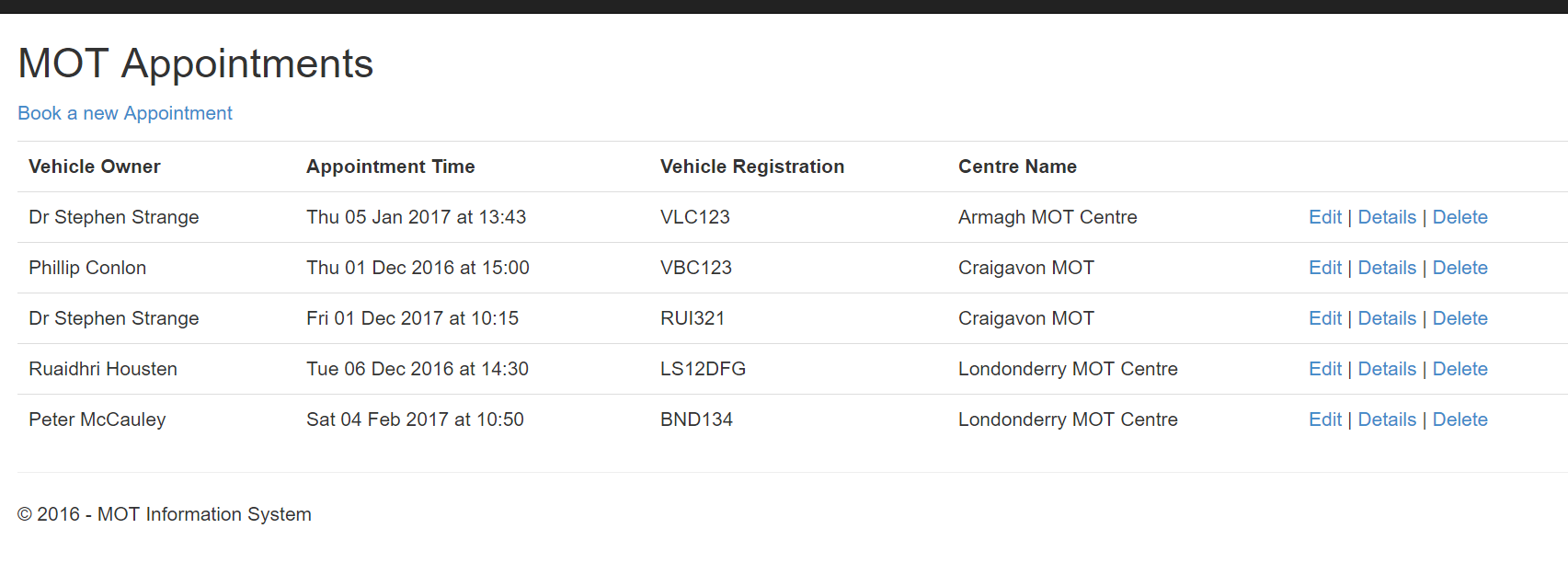
### Deleting an Appointment



To delete an appointment, the user can click the Delete link to the right of the record. All dates use the date format from the Appointment.cs file. This gives a detailed description for the date of the appointment. Shown below is the table after the record has been deleted.

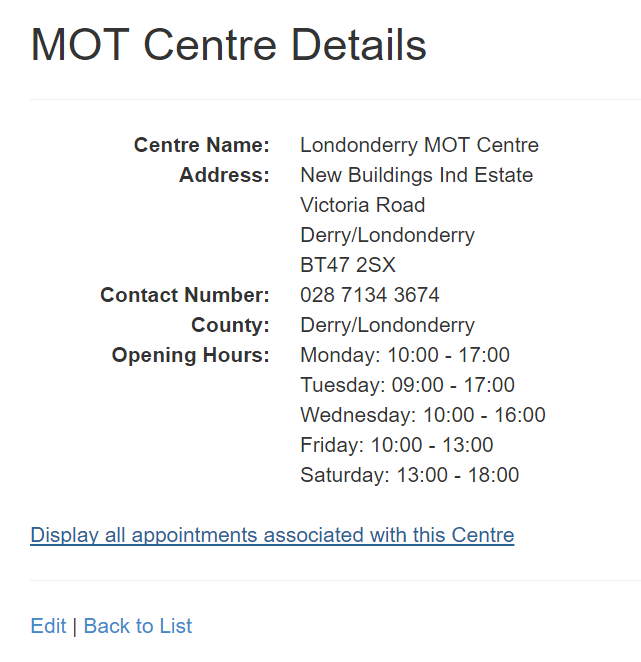


### List of all Appointments



Shown here is a list of all the appointments using the formatting supplied in the Appointment.cs file. The user has the options to Book New Appointments; Edit, Detail or Delete existing appointments.

### List of a Centre’s Appointments



To list the Appointments associated with a Centre the user must navigate to the specific Centre’s Details page. From there then can select the link which will display the appointmests.

