Sistema gestor de citas:

Se realizará mediante el actor <<Trabajador>>

Requisitos funcionales:

RF-01: Autenticación/Loggin de Trabajador.
RF-02: Crear cita seleccionando un inspector y añadir datos del vehículo y su propietario.
RF-03: Eliminar cita de la base de datos.
RF-04: Actualizar cita de la base de datos.
RF-05: Exportar cita + informe de la cita.
RF-06: Exportar todas las citas + informes a JSON.

Requisitos no funcionales:

RNF-01.1: La creación de una cita se realiza mediante un inspector con menos de 5 citas asociadas, y se deberán introducir los datos del vehículo y su propietario, generándose un informe el cual estará vacío con al atributo “no apto”.
RNF-01.2: Una cita se podrá crear si hay menos de 8 citas creadas en el intervalo de 30 minutos.
RNF-02: La búsqueda de la cita se podrá realizar mediante la matrícula del vehículo, tipo de vehículo y la fecha de la última revisión.
RNF-03: Al actualizar una cita, se selecciona si es apto o no apto, asignando los datos del informe, y actualizando el informe que se encuentra vacío.
RNF-04: Al exportar un informe de la cita, podremos elegir el formato de exportación a JSON o Markdown, tendrá los datos del propietario del vehículo, el vehículo, el trabajador que gestione la inspección, datos del informe de la inspección y datos de la cita.
RNF-05: La base de datos está creada en MariaDB.
RNF-06: La lógica de la aplicación está realizada con Kotlin.

Requisitos de información:

RI-01: El ID será la clave primaria.
RI-02: El estado corresponderá a “Apto” o “No apto”.
RI-03: La fecha y hora tendrá formato: yyyy-MM-ddTHH:mm:ss.SSS
RI-04: Tendrá un ID de informe que hará referencia a los datos siguientes:
RI-04.1: Aptitud del frenado del vehículo, decimal formato: 0.00
RI-04.2: Aptitud de la contaminación del vehículo, decimal formato: 0.00
RI-04.3: La fecha del informe tendrá un formato: yyyy-MM-dd
RI-04.4: Aptitud del interior del vehículo, mediante apto o no apto.
RI-04.5: Aptitud de luces del vehículo, mediante apto o no apto.
RI-04.5: Aptitud general del vehículo, mediante apto o no apto.
RI-05: Tendrá como referencia al usuario del trabajador que gestionará la cita.
RI-06: Tendrá como referencia a la matrícula del vehículo que se realizará la inspección

Sistema gestor de trabajadores:

Se realizará mediante el acto <<Admin>>

Requisitos funcionales:

RF-07: Autenticación/Loggin de Admin.
RF-08: Guardar trabajador en la base de datos.
RF-09: Eliminar trabajador de la base de datos.
RF-10: Actualizar trabajador en la base de datos.
RF-11: Realizar trabajador del inspector.
RF-12: Exportar todos los trabajadores a CSV

Requisitos no funcionales:

RNF-07: La búsqueda se realizará mediante el DNI.
RNF-08: La base de datos está creada en MariaDB.
RNF-09: La lógica de la aplicación está realizada con Kotlin.

Requisitos de información:

RI-07: El trabajador como clave primaria será el nombre de usuario y será único.
RI-08: El trabajador en el email será único.
RI-09: El trabajador tendrá una contraseña cifrada.
RI-10: El trabajador tendrá la fecha de contratación en formato yyyy-mm-dd.
RI-11: El trabajador en función de su especialidad tendrá un salario fijo, en caso de tener más de tres años de antigüedad tendrá un bonus de +100€.
RI-11.1: Administración (1650€)
RI-11.2: Electricidad (1800€)
RI-11.3: Motor (1700€)
RI-11.4: Mecánica (1600€)
RI-11.4: Interior (1750€)
RI-11.5: Responsable (+1000€ sobre el salario de la especialidad que tenga)
RI-12 Cada trabajador no atenderá más de 4 citas por intervalo de 30 minutos.
RI-13 El trabajador tendrá como referencia el ID de la estación en la que trabaja.
RI-14 El trabajador tendrá un campo para saber si es el gerente de la estación o no.

Sistema gestor de vehículos:

Se realizará mediante el acto <<Admin>>

Requisitos funcionales:

RF-13: Autenticación/Loggin de Admin.
RF-14: Crear un vehículo.
RF-15: Actualizar vehículo.
RF-16: Eliminar vehículo de la base de datos.

Requisitos no funcionales:

RNF-10: La búsqueda se realizará mediante la matrícula.
RNF-11: La base de datos está creada en MariaDB.
RNF-12: La lógica de la aplicación está realizada con Kotlin.

Requisitos de información:

RI-15: La matriculación debe ser una cadena que favorezca la expresión “1111XXX”.
RI-16: La marca debe ser una cadena.
RI-17: El modelo debe ser una cadena.
RI-18: La fecha de matriculación debe tener formato yyyy-mm-dd
RI-19: La fecha de revisión debe tener formato yyyy-mm-dd
RI-20: El tipo de motor podrá ser: gasolina, diésel, híbrido o eléctrico.
RI-21: El tipo de vehículo podrá ser: turismo, furgoneta, camión o motocicleta.
RI-22: El DNI del propietario será una cadena como referenciada del propietario.

Sistema gestor de propietarios:

Se realizará mediante el acto <<Admin>>

Requisitos funcionales:

RF-17: Autenticación/Loggin de Admin.
RF-18: Crear un propietario.
RF-19: Actualizar propietario.
RF-20: Eliminar propietario de la base de datos.

Requisitos no funcionales:

RNF-13: La búsqueda se realizará mediante el DNI.
RNF-14: La base de datos está creada en MariaDB.
RNF-15: La lógica de la aplicación está realizada con Kotlin.

Requisitos de información:

RI-23: El DNI será una cadena clave primaria.
RI-24: El nombre será una cadena.
RI-25: El apellido será una cadena.
RI-26: El correo será una cadena con el formato: xxx@xxx.xx
RI-27: El teléfono será una cadena con el formato: 666 666 666

Sistema gestor de estaciones:

Se realizará mediante el acto <<Super-Admin>>

Requisitos funcionales:

RF-21: Autenticación/Loggin de Super-Admin
RF-22: Guardar estación en la base de datos.
RF-23: Actualizar estación en la base de datos.
RF-24: Eliminar estación de la base de datos.

Requisitos no funcionales:

RNF-16: La búsqueda se realizará mediante el ID.
RNF-17: La base de datos está creada en MariaDB.
RNF-18: La lógica de la aplicación está realizada con Kotlin.

Requisitos de información:

RI-28: El ID es una clave primaria.
RI-29: El nombre es una cadena.
RI-30: La dirección es una cadena.
RI-31: El correo tiene un formato: xxx@xxx.xx
RI-32: El teléfono es una cadena con formato: 666 666 666