**1º Realiza un servicio con las siguientes rutas y funcionalidades dentro del mismo servicio:**

**Fechas a partir de números.**

La ruta admitirá un número comprendido entre 1 y 365 y calculará el mes y el día del mes que corresponde (año no bisiesto). Por ejemplo, para el número 75, se **devolverá el día 16 del mes 3, en formato JSON.**

Se controlará el error de que el número proporcionado no esté comprendido entre 1 y 365.

**Complemento a 1.**

**La ruta admitirá un número binario, y lo pasará a complemento a 1, devolviendo el resultado en JSON**. Asimismo, debemos **diseñar lo necesario** para que compruebe si el número introducido **es válido o no** (sólo tiene dígitos 1 y 0) y **se realice la conversión**. Si el número no fuera válido se informará debidamente al cliente.

**Nota:** El complemento a uno de un número binario es una operación matemática que nos permite la representación binaria de números negativos. Se obtiene al cambiar cada uno de los dígitos del número binario *N* por su complementario, esto es, cambiar los unos por ceros y los ceros por unos

Número binario = (01010110)2 = (86)10

Complemento a uno = (10101001)2 = ( − 86)10

Opción 2: Que la ruta permita también realizar el complemento a 2. Se deja al programador las posibles rutas necesarias o métodos de proporcionar los datos necesarios: número y conversión deseada.

**Alea Iacta Est.**

Son famosas las formaciones que el antiguo ejército romano utilizaba para entrar en batalla. En esas formaciones, los legionarios se agrupaban en una figura geométrica (normalmente un rectángulo) y protegían **tanto los flancos como la parte superior utilizando escudos.** Los legionarios que ocupaban **posiciones interiores cubrían la parte superior colocando el escudo sobre su cabeza**, mientras que **los que ocupaban los flancos llevaban dos y hasta tres escudos**: uno para proteger la parte superior y uno o dos escudos (si estaban en la esquina) para proteger los laterales. En esta formación todos los legionarios quedaban protegidos por los escudos y eran muy difíciles de vencer.

Cuenta la historia que existió un general que estableció que la mejor figura para la formación no era la rectangular sino la **cuadrada**, de forma que el número de filas y columnas de legionarios coincidía.

El problema al que se enfrentaba este general era decidir en cuántas formaciones (y de qué tamaño) debía separar su ejército para que:

* No quedará **ningún legionario fuera** de una formación (aunque admitía formaciones de un único legionario; en este caso ese legionario debía portar 5 escudos!!!!).
* Se minimizará el número de escudos necesarios para protegerlos.

Nuestro general, después de hacer muchos cálculos, decidió que la mejor manera de que estas dos condiciones se cumpliesen era **comenzar haciendo el cuadrado más grande posible con sus legionarios. Con los que le quedasen libres volvía a repetir la operación, y así hasta que no quedasen legionarios que formar**.

Por ejemplo, si el número de legionarios en el ejército era 35, la manera utilizada por el general para hacer la formación consistía en un cuadrado de 25 legionarios (5x5), otro de 9 (3x3) y otro de 1 (1x1):



Esta formación requería un total de 71 escudos.

Se pide una **ruta que admita el número total de legionarios** y nos **calcule** **los cuadrados que hay que ir haciendo con su dimensión**, así como el **número total de escudos** necesarios en total. Toda esta información será devuelta al cliente en el formato adecuado.

**2º MasterMind.**

Escribe un **servicio API Rest en NodeJS** que juegue al **Rojo – Amarillo – Verde**. El programa **genera tres dígitos aleatorios distintos entre 0 y 9**. A estos dígitos se les asignan las posiciones 1, 2 y 3. **El objetivo del juego es adivinar los números y sus posiciones correctas en el menor número de intentos posibles**. Para **cada intento** el jugador proporciona tres números para las posiciones 1, 2 y 3. El programa responde con una **pista** que consta de rojo, amarillo y verde. Si un dígito está en la posición correcta, la respuesta es *verde (v),* si el digito adivinado está en una posición incorrecta, la respuesta es *amarillo (a)* y si el dígito no coincide con ninguno de los tres dígitos la respuesta es *rojo (r)*.

**Ejemplo:** 6 5 8 es la combinación a averiguar.

Intentos: 1 2 5

Pistas: r r a

Intentos: 8 5 3

Pistas: a v r

Intentos: 8 5 6

Pistas: a v a

**Se deja a criterio del programador o programatriz rutas y número de usuarios/partidas abiertas que se permiten.**

**3º CRUD de tareas.**

Investiga un equivalente al Faker de Laravel para NodeJS. Al arrancar el servidor se llenará una colección de personas con datos aleatorios. Proporciona las rutas necesarias para la funcionalidad clásica de un CRUD de tareas. Estas rutas deben estar agrupadas bajo ‘/tasks.

Realiza, asimismo, un login y un registro.

Las tareas tendrán: descripción, duración estimada (horas), dificultad (XS, S, M, L, XL) y si está realizada o no.

Cuando todo funcione, añade protección al CRUD para que solo los administradores puedan hacer uso de sus rutas.

Se valora que el diseño sea bien organizado, robusto, preciso y claro.

**4º Umbrella Corporation.**

Nos ha contratado la empresa de Resident Evil. Vamos a programar algunas cosas para la supercomputadora que gobierna La Colmena: **La Reina Roja.**

Tendremos que diseñar las siguientes rutas con sus respectivas funcionalidades:

***Abrir\_Puertas(sala):*** Ruta que abre todas las puertas de la sala que le pasemos.

***Cerrar\_Puertas(sala):*** Ruta que cierra todas las puertas de la sala que le pasemos.

***Puertas\_Abiertas(sala):*** Ruta que me devolverá si las puertas de esa sala están abiertas o no.

***Detectar(tipo, sala):*** Ruta que según el tipo que le pasemos (‘h’ o ‘z’) y el número de sala devolverá cuantos humanos o zombis hay en esa sala.

***Aumentar\_Virus(sala):*** Ruta que aumenta la densidad del virus en el aire en 0,1.

***Disminuir\_Virus(sala):*** Ruta que disminuye la densidad del virus en el aire en 0,1.

***Densidad\_Virus(sala, densid):*** Ruta que me devuelve la densidad de virus que hay en la sala.

Se deja a criterio de los programadores y/o/u programatrices la elección de verbos y rutas.

Por otro lado, La Reina Roja ha detectado que un comando de hombres ha entrado en La Colmena y como se ha vuelto homicida trata de cargárselos. **En todas las salas habrá zombis y puede que humanos.**

**Tenemos que diseñar un servicio para que La Reina Roja pueda comprobar el estado de las 9 salas.**

Para cada sala se hace lo siguiente:

- si hay sólo zombis abre las puertas (si no están abiertas).

- si hay algún humano, cierra las puertas y aumenta el nivel de intensidad del virus hasta el límite 20 para infectar a los humanos. Permanece comprobando la sala hasta que los humanos que había en la sala sean zombis (sólo se detectarán zombis, por lo tanto). Cuando todos sean zombis se disminuye la intensidad del virus en el ambiente hasta la cantidad que la sala tenía originalmente y después se abren las puertas.

**El programa acabará cuando en todas las salas sólo haya zombis.**

En este ejercicio debes decidir cómo usa la Reina Roja los servicios definidos para realizar en el cliente el programa indicado.