**Type: E  
Points: 1  
1. Explica con tus palabras el funcionamiento del Loop de Eventos de Node.**

El Loop de Eventos de Node.js es un bucle continuo que ejecuta eventos de manera asíncrona.

Por ejemplo, tenemos una lista de tareas para hacer, pero algunas de ellas llevan mucho tiempo. En lugar de esperar a que una tarea termine antes de pasar a la siguiente, el Loop de Eventos revisa si hay alguna tarea pendiente. Si hay alguna tarea que se puede realizar rápidamentre y que sea sincrónica, la realiza de inmediato, si la siguiente tarea requiere mucho tiempo o es asíncrona, en lugar de esperar a que se complete, Node.js la delega a los componentes internos del sistema operativo y continúa con la siguiente tarea. Cuando una tarea asíncrona termina, el Loop de Eventos ejecuta la parte del código que estaba esperando ese resultado.

Resumidamente, el Loop de Eventos de Node.js permite que el código se ejecute de forma más eficiente y sin bloqueos.

**Type: E  
Points: 1  
2. Enumera 4 módulos de node.js y explica brevemente para que sirven.**

1. Express. Se utiliza para crear aplicaciones web y APIs de forma sencilla, facilita la creación de servidores web.
2. fs – Permite utilizar archivos que tengamos en el ordenador (leer, escribir, modificar y eliminar archivos)
3. http. Permite crear servidores y procesar solicitudes de clientes

path. Proporciona funciones para trabajar con rutas a archivos y directorios

**Type: E  
Points: 1  
3. ¿Cuál es la diferencia entre un viewset y un api\_view? ¿Cuál usarías para desarrollar una API simple de forma rápida?  
Justifica la respuesta**

ViewSet: ViewSet es una clase que combina múltiples operaciones (Crear, Leer, Actualizar, Eliminar)

APIView: APIView es una clase que proporciona métodos para manejar solicitudes HTTP específicas (GET, POST, PUT, DELETE). Se deben definir manualmente los métodos para cada acción HTTP admitida en el API

Para desarrollar una API simple de forma rápida es más conveniente utilizar un ViewSet, porque agrupa las operaciones comunes y es más rápido de implementar.

**Type: E Points: 3 4. Desarrolla una API simple que nos permita tener una lista de tareas (añadir, actualizar, borrar y listar) con el stack tecnológico que prefieras. (subir código y breve explicación de como se ha realizado)**

const express = require('express');

const mongoose = require('mongoose');

const app = express();

const PORT = 3000;

app.use(express.json());

mongoose.connect('mongodb+srv://verasavinave:AzrPqownGJYq3Cqc@cluster0.uykfgh9.mongodb.net/tareas?retryWrites=true&w=majority')

.then(() => console.log('Connected to Mondo'))

.catch((err) => console.log('Error connecting to MongoDB', err));

const tareasSchema = new mongoose.Schema({

    titulo: {type: String, required: true},

    descripcion: String,

    status: { type: String, default: 'to-do' },

});

const Tarea = mongoose.model('Tarea', tareasSchema);

app.post('/tareas', async (req, res) => {

    try {

    const tarea = new Tarea(req.body);

    await tarea.save();

    res.status(201).send(tarea);

} catch(error) {

    res.status(400).send({error: error.message});

}

});

app.get('/tareas', async (req, res) => {

    try {

    const tareas = await Tarea.find();

    res.send(tareas);

} catch(error) {

    res.status(500).send({error: error.message});

}

});

app.delete('/tareas/:id', async (req, res) => {

    try {

    const tareas = await Tarea.findByIdAndDelete(req.params.id);

    if (!tarea) {

        return res.status(404).send({error: 'Tarea no encontrada'});

    }

    res.send(tarea);

} catch(error) {

    res.status(500).send({error: error.message});

}

});

app.put('/tareas/:id', async (req, res) => {

    try {

    const tarea = await Tarea.findByIdAndUpdate(req.params.id, req.body);

    if (!tarea) {

        return res.status(404).send({error: 'Tarea no encontrada'});

    }

    const new\_tarea = await Tarea.findById(req.params.id);

    res.send(new\_tarea);

} catch(error) {

    res.status(500).send({error: error.message});

}

});

app.listen(PORT, () => {

    console.log(`Server listening on port ${PORT}`);

  });

**Type: E Points: 1 5. ¿Qué es el backend?**

El backend se encarga de proporcionar la funcionalidad de API, disponible para los usuarios a través de la interfaz de frontend. Interactúa con la base de datos, gestiona usuarios y sesiones, maneja los archivos y otras tareas relacionadas con el procesamiento y almacenamiento de datos

**Type: E Points: 1 6. ¿Por qué has elegido el stack utilizado en la pregunta 4? Elabora tu respuesta**

Como de momento no tengo ninguna preferencia y en las últimos clases hemos utilizado Django, quería hacer un ejercicio de repaso de Node.js

**Type: E Points: 1 6. Escribe el código minimo necesario del fichero models.py para crear un modelo ‘Tienda’, que cuente con un nombre, una dirección, una ciudad, un código postal y una fecha de apertura. Define también una propiedad devuelva un entero con los días que lleva.**

from django.db import models

from datetime import datetime

class Tienda (models.Model):

    name = models.CharField(max\_length=100)

    address = models.TextField()

    city = models.CharField(max\_length=100)

    postcode = models.IntegerField()

    opendate = models.DateField()

    @property

    def days\_opened(self):

        days = (datetime.now().date() - self.opendate).days

        return days

**Type: E Points: 1 7. ¿Qué es un serializador? Escribe el código necesario para crear un serializador (serializer.py) para el modelo anterior. Añade también un comprobante que solo nos permita añadir tiendas que estén en Madrid.**

Un serializador en es una clase que permite convertir los modelos de base de datos, en representaciones más simples, como JSON y validar los datos untroducidos.

from rest\_framework import serializers

from .models import Tienda

class UserSerializer(serializers.ModelSerializer):

    class Meta:

        model = Tienda

        fields = ['name', 'address', 'city', 'postcode', 'opendate']

    def validate\_city(self, city):

        if city != 'Madrid':

            raise serializers.ValidationError("La tienda debe estar en Madrid.")

        return city

be el código necesario