



**Clase 38.** Programación Backend

# ***Arquitectura de capas***



## ***OBJETIVOS DE LA CLASE***

- Aprender acerca de npm más en profundidad.
- Comprender la configuración de un proyecto en capas.

# ***CRONOGRAMA DEL CURSO***

Clase 37



**Versiones y paquetes**

Clase 38



**Arquitectura de capas**

Clase 39



# ***IMPLEMENTACIÓN DE PROYECTOS EN CAPAS***

# ***¿Qué es la arquitectura de capas?***



- Es un patrón que se utiliza en desarrollo de software donde los roles y responsabilidades dentro de la aplicación (app) son separadas en capas.
- Uno de sus objetivos es separar responsabilidades entre componentes.
- Otro objetivo es organizar las capas para que ellas lleven a cabo su labor específica dentro del app.
- Una app pequeña suele contener 3 capas. Luego, el número de capas puede ir aumentando con la complejización de la app.

# ***Capas principales en Node***



Las 3 capas principales que tienen también las app pequeñas son:

1

**Capa de ruteo:** maneja la interfaz de programación de aplicaciones (**API**). Su único trabajo es recibir las peticiones del cliente, delegar la tarea de computar la respuesta, y una vez obtenido el resultado retornarlo como respuesta al cliente.

# *Capas principales en Node*



- 2 **Capa de servicio:** maneja la lógica de negocios del app. Significa que los datos son transformados o calculados para cumplir con los requerimientos del cliente. Accede a los datos (leer-guardar) sólo a través de la capa de persistencia.

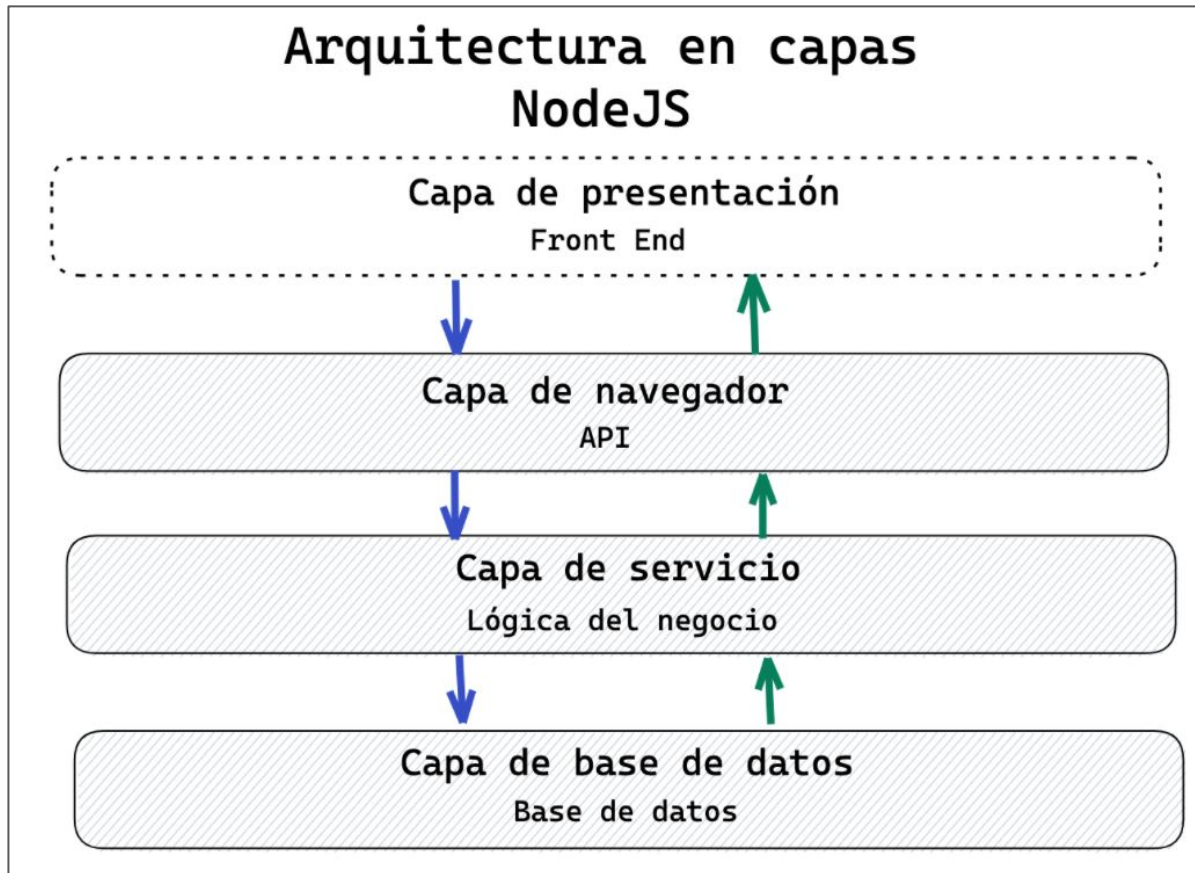
# *Capas principales en Node*



- 3 **Capa de persistencia:** tiene acceso a la base de datos para crear, editar, o borrar datos. Preferentemente, aquí no debemos encontrar lógica de negocio, sino mecanismos relacionados con la infraestructura del servidor.



# Diagrama de arquitectura de capas



# ***Movimiento entre capas***



- La travesía de los datos comienza en la **capa de presentación** donde el usuario hace click y llama a la función que envía la solicitud de datos a la API, representada en nuestro sistema por la capa de ruteo.
- El componente en la capa de ruteo llama al componente en la capa de servicio, y se encarga de esperar por la respuesta de la capa de servicio para así retornarlo.

# ***Movimiento entre capas***



- Los datos son transformados o calculados en la capa de servicio antes de devolver una respuesta (o de pasarlos a la siguiente capa).
- En este último caso, el componente en la capa de servicios llama al componente inyectado de la capa de persistencia y le pasa los datos.

# ***Movimiento entre capas***



- Finalmente, se lleva a cabo la solicitud de datos al servidor de base de datos en la capa de persistencia. A esta capa se accede mediante llamados asincrónicos (preferentemente utilizando promesas).
- Cuando el llamado a la capa de persistencia se resuelve con la respuesta del servidor de base de datos, la respuesta retorna a la capa de servicio. Esta retorna a la capa de rutas. Cuando la respuesta alcanza la *capa de rutas*, los datos llegan al usuario a través de la *capa de presentación*.
- Debemos tener en cuenta que este siempre es el camino, **los datos, solicitudes y respuestas no saltean capas.**

# ***IMPLEMENTACIÓN DE ARQUITECTURA DE CAPAS***

# ***Implementar la arquitectura de capas***



Como ya mencionamos, es muy importante **separar responsabilidades** entre componentes. Para nuestra separación utilizaremos 5 componentes:

- Server: inicializa la aplicación, y carga los routers correspondientes.
- Router: que contiene los métodos disponibles para cada recurso de la aplicación, y sendas mediante las cuales se los accede.
- Controller: contiene las funciones que resolverán cada petición que llegue a cada una de las rutas definidas.
- Service: contiene las funciones con la lógica de negocio relacionada a los recursos del sistema.
- Data Access Object (DAO): contiene las funciones relacionadas con el acceso a la base de datos.

# Implementar la arquitectura de capas



- Server es la cara visible de nuestro servidor, la *interfaz* entre el Cliente y la *capa de ruteo* del servidor.
- Los componentes de ruteo representan la **Capa de Ruteo**, y son los únicos en donde encontraremos referencias a la naturaleza de cliente-servidor de nuestro sistema (por ejemplo: referencias a la librería Express).
- Los controladores constituyen la *interfaz* entre la *capa de ruteo* y la *capa de negocios*.
- Los componentes de servicios representa nuestra **Capa de Negocios**. Aquí se realizan todas las validaciones y se toman todas las decisiones para cuidar que se cumplan las reglas del negocio que se está modelando. Estos componentes generalmente no contienen referencias a tecnologías específicas ni frameworks utilizados, únicamente javascript puro.
- Los *DAOs* (data access objects: objetos de acceso a datos) constituyen la **Capa de Persistencia**, y son los únicos lugares en donde encontraremos referencias al almacenamiento de datos (ejemplo: referencias a la librería Mongoose, o Firebase, etc).

# *Capa de persistencia*



```
const datos = []

async function recuperarTodos() {
  return datos
}

async function guardar(dato) {
  datos.push(dato)
  return dato
}

export {
  recuperarTodos,
  guardar
}
```



# Capa de servicio



```
import { recuperarTodos, guardar } from '../persistencia/datos.js';

async function obtenerDatos() {
  return await recuperarTodos()
}

async function crearDato(dato) {
  dato.added = Date.now()
  await guardar(dato)
  return dato
}

export { obtenerDatos, crearDato }
```

# Controllers



```
import { obtenerDatos, crearDato } from '../negocio/datos.js'

async function getDatosController(req, res) {
  const datos = await obtenerDatos()
  res.json(datos)
}

async function postDatosController(req, res) {
  const dato = req.body
  await crearDato(dato)
  res.status(201).json(dato)
}

export { getDatosController, postDatosController }
```

# Capa de ruteo



```
import { Router } from 'express'
import { getDatosController, postDatosController } from '../controlador/datos.js'

const routerDatos = new Router()

routerDatos.get('/', getDatosController)
routerDatos.post('/', postDatosController)

export default routerDatos
```

# ***Servidor***

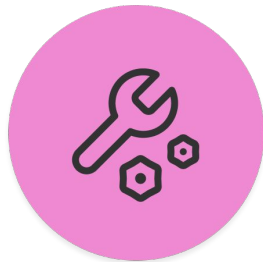


```
import express from 'express'
import routerDatos from './rutas/datos.js'

const app = express()
app.use(express.json())

app.use('/api/datos', routerDatos)

// start server
const PORT = 8080
const server = app.listen(PORT, () => {
  console.log(`Servidor express escuchando en el puerto ${server.address().port}`)
})
server.on('error', error => console.error(`Error en servidor`, error))
```



# ***SERVIDOR EN CAPAS***

*Tiempo: 15 minutos*

# ***Servidor en capas***

*Tiempo: 15 minutos*

Desafío  
generico



Dividir en capas el servidor del último desafío entregable, separando en capas de ruteo/controlador, lógica de negocio y capa de persistencia, utilizando exclusivamente funciones como vimos en el ejemplo de la clase.

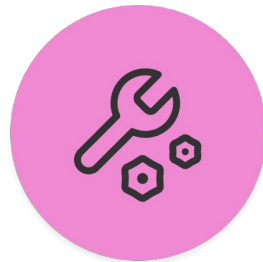
Además de ubicar en cada una de esas capas lo realizado hasta el momento, guardar en persistencia las operaciones realizadas en un array de objetos que contengan: tipo de operación, parámetros, resultado y timestamp.

Permitir además el listado de dichas operaciones registradas.



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**



# ***SERVIDOR EN CAPAS (2da parte)***

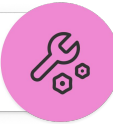
*Tiempo: 10 minutos*



# Agregar capas al servidor

Tiempo: 10 minutos

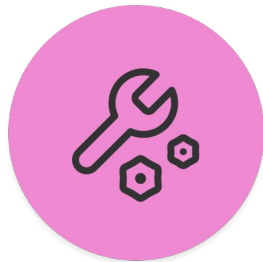
Desafío  
generico



Continuando el proyecto del desafío anterior, modificar nuestro sistema para que la persistencia, en lugar de realizarse en memoria, se realice en el disco, en un archivo.

Luego, modificar nuevamente el sistema para que la persistencia se realice utilizando una base de datos (a elección del desarrollador).

De ser posible, realizar ambas modificaciones sin tener que tocar la capa de negocios.



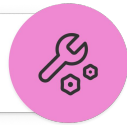
# ***SERVIDOR EN CAPAS (3ra parte)***

*Tiempo: 10 minutos*

# Agregar capas al servidor

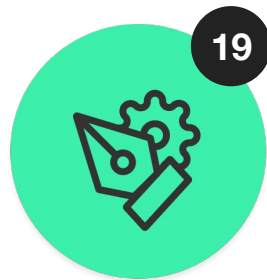
Tiempo: 10 minutos

Desafío  
generico



Continuando el proyecto del desafío anterior, modificar nuestro sistema para que la operacion de 'listar' solo pueda ser realizada por usuarios autorizados. Para ello, agregar un mecanismo simple de registro, que provea una clave secreta, y que solo permita acceder al listado de operaciones a quienes adjunten a su peticion la clave en su cabecera.

De ser posible, realizar estas modificaciones sin tener que tocar la capa de negocios ni la de persistencia.



# ***DIVIDIR EN CAPAS NUESTRO PROYECTO***

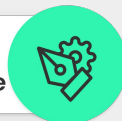
Retomemos nuestro trabajo para dividir en capas el proyecto como aprendimos.

# ***DIVIDIR EN CAPAS NUESTRO PROYECTO***

**Formato:** link a un repositorio en Github con el proyecto cargado.

**Sugerencia:** no incluir los node\_modules

Desafío  
entregable



**>> Consigna:** Dividir en capas el proyecto entregable con el que venimos trabajando (entregable clase 16: loggers y profilers), agrupando apropiadamente las capas de ruteo, controlador, lógica de negocio y persistencia.

Considerar agrupar las rutas por funcionalidad, con sus controladores, lógica de negocio con los casos de uso, y capa de persistencia.

La capa de persistencia contendrá los métodos necesarios para atender la interacción de la lógica de negocio con los propios datos.

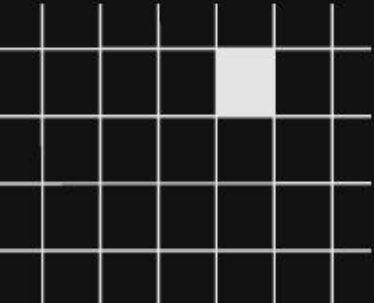
***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Administradores de paquetes de Node.
  - Npm, usos y aplicaciones. Subir nuestro propio paquete.
  - Nvm, qué es y cómo usarlo.
  - Arquitectura de proyectos en capas.
- 



***OPINA Y VALORA ESTA CLASE***



***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODER HOUSE***