

# **Curso de Backend con Node.js**





# Nicolas Molina

Google Developer Expert  
Dev and Teacher at Platzi  
@nicobytes

**¿Qué es  
Express.js?**

Search product

Order: Most recent



\$ 35,00  
Round shelf



\$ 120,00  
Retro refrigerator



\$ 35,00  
Round shelf



\$ 120,00  
Retro refrigerator



\$ 35,00  
Round shelf



```
const express = require('express');  
const app = express();  
const port = 3000;
```

```
app.get('/', (req, res) => {  
  res.send('Hello World!')  
});
```

```
app.listen(port, () => {  
  console.log(`http://localhost:${port}`)  
});
```

# Configuración

# **Tu primer server**

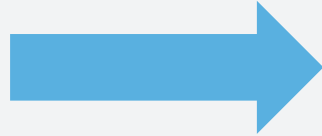
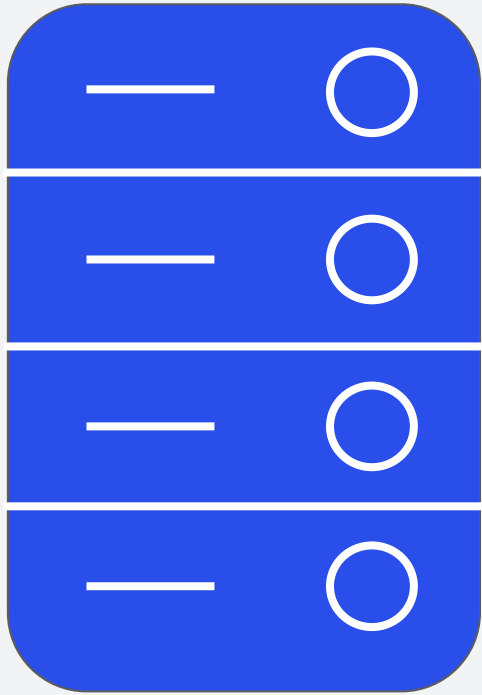


# Routing

# API Restful

# REST

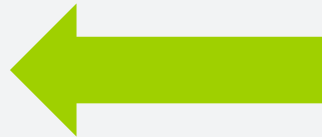
Representational State Transfer



**GET**



**PUT**





**POST**



**DELETE**



Method	/products	/products/{id}
GET	Get list	Get
PUT	Replace 	Update/Replace
PATCH	No Apply	Update
POST	Create	No Apply
DELETE	Delete 	Delete

**GET**

api.example.com/tasks/{id}/  
api.example.com/people/{id}/  
api.example.com/users/{id}/tasks/

# Query Params



api.example.com/products

api.example.com/products?page=1

api.example.com/products?limit=10&offset=0

api.example.com/products?region=USA

api.example.com/products?region=USA&brand=XYZ

# **Single Responsibility Principle**



**POST**

api.example.com/api/v1/users/

api.example.com/api/v1/tasks/

api.example.com/api/v1/customers/

**DELETE**  
**PUT**  
**PATCH**

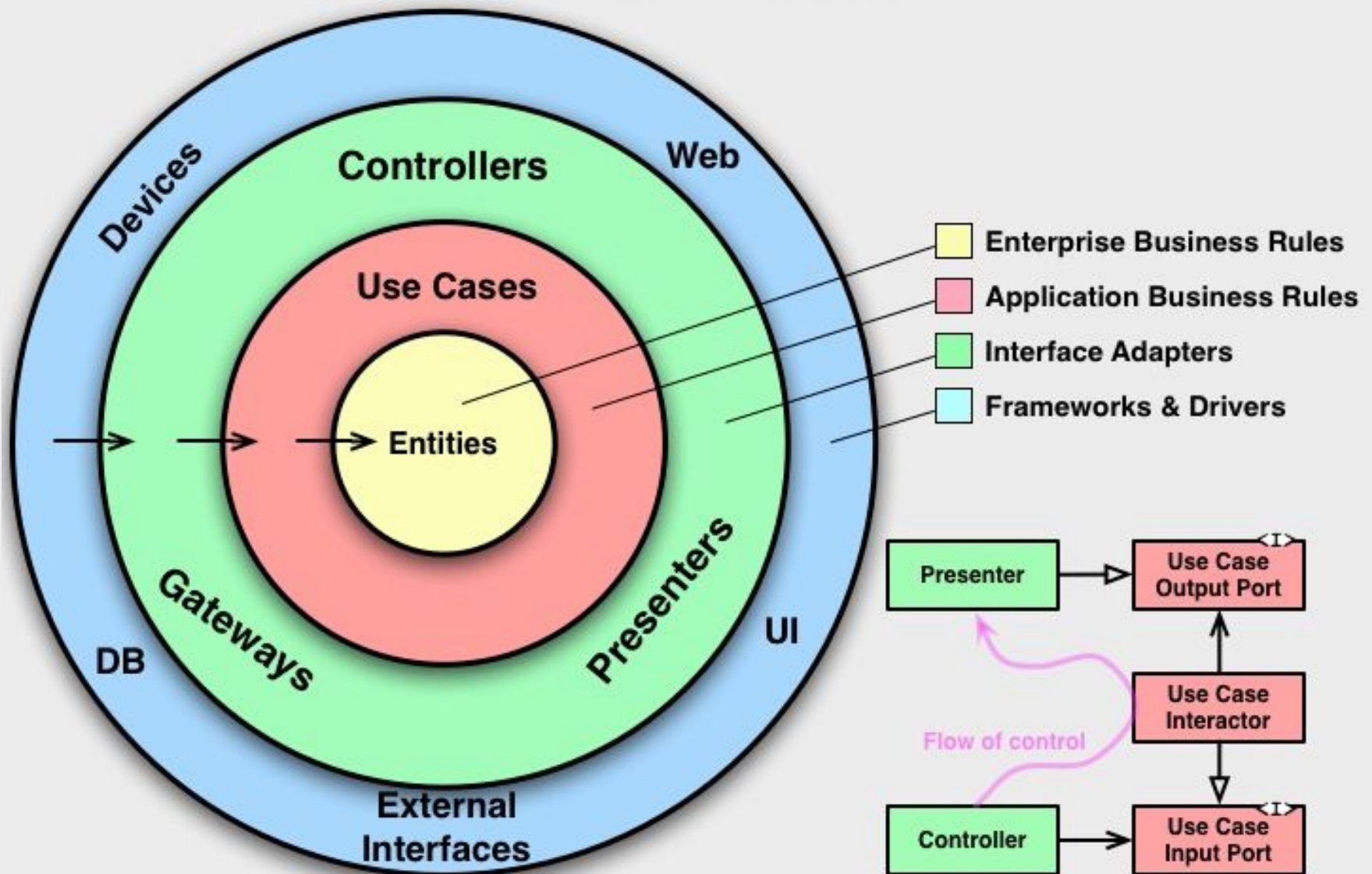
Method	/products	/products/{id}
GET	Get list	Get
PUT	Replace 	Update/Replace
PATCH	No Apply	Update
POST	Create	No Apply
DELETE	Delete 	Delete

**STATUS CODE**

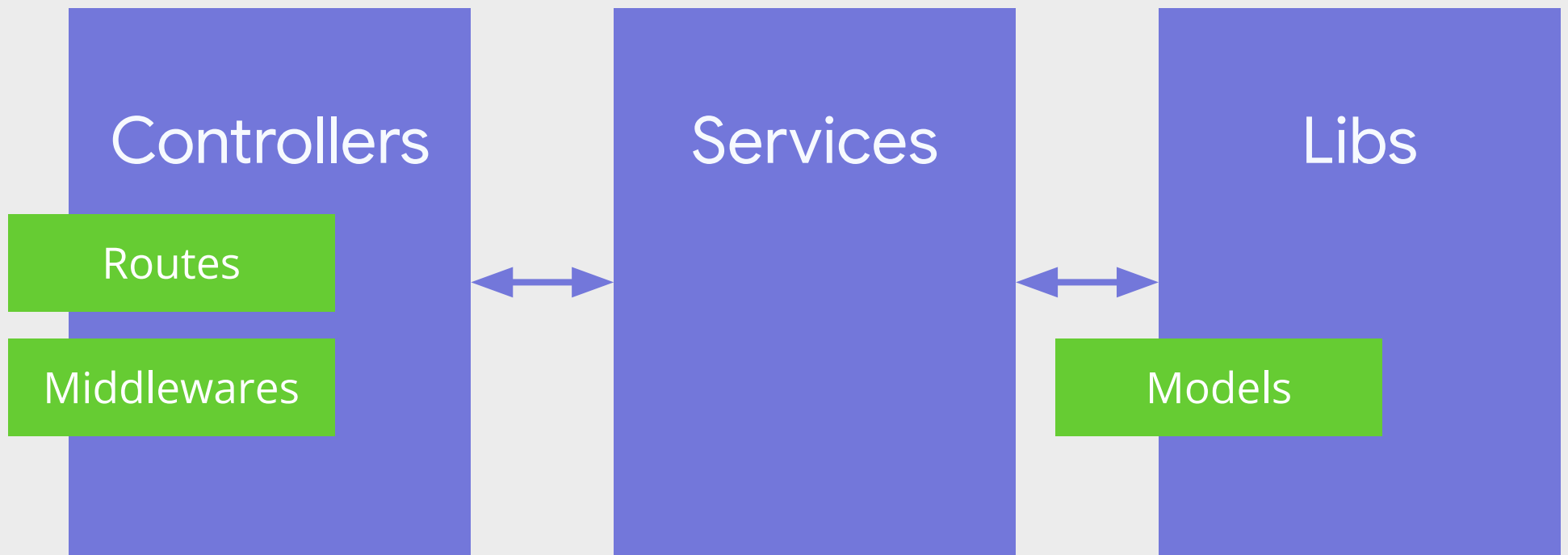
**SERVICIOS**



# The Clean Architecture



# The Clean Architecture



**Create,  
update & delete**

# Async

# Middlewares





```
graph LR; A[Middleware] --> B[Middleware]; B --> C[Middleware];
```

Middleware

Middleware

Middleware

```
graph LR; M1[Middleware] --> M2[Middleware]; M3[Middleware];
```

Middleware

Middleware

Middleware



```
function (req, res, next) {  
  if (something) {  
    res.send( 'end' );  
  } else {  
    next();  
  }  
}
```

```
function (error, req, res, next) {  
  if (error) {  
    res.status(500).json({error});  
  } else {  
    next();  
  }  
}
```

# Uses Cases

- Funcionar como pipes
- Validar datos
- Capturar errores
- Validar permisos
- Controlar accesos

# Middleware for Errors

# **Manejo de errores con Boom**

# **Validación de datos con Joi**

# **Probando los endpoints**

# **Consideraciones para producción**



# Recomendaciones

- Cors
- Https
- Procesos de Build
- Remover logs
- Seguridad (Helmet)
- Testing

api.mydomain.com



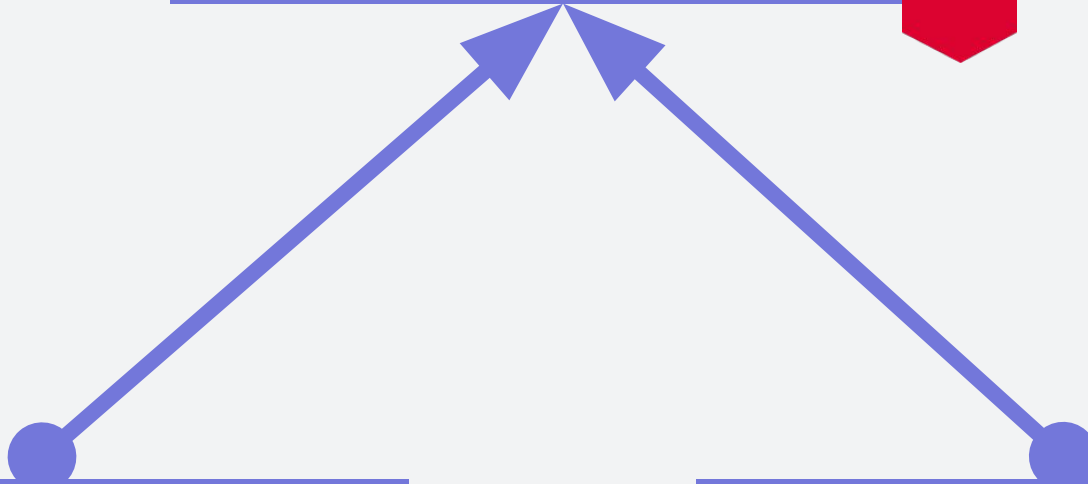
api.mydomain.com

api.mydomain.com



mydomain.com

otro.mydomain.com



**CORS**

# Deployment to Heroku

```
curl https://cli-  
assets.heroku.com/install.sh | sh
```

```
heroku login  
heroku create  
git remote -v
```

# Deployment to Heroku

heroku logs --tail



```
const express = require('express');  
const app = express();  
const port = 3000;
```

```
app.get('/', (req, res) => {  
  res.send('Hello World!')  
});
```

```
app.listen(port, () => {  
  console.log(`http://localhost:${port}`)  
});
```