Explicación del Servidor Express + MongoDB

🏗️ Estructura del Servidor (Backend)

1. **Importaciones y Configuración**

```
import express from 'express'
import mongoose from 'mongoose'
import cors from 'cors'
import dotenv from 'dotenv'
import Task from './models/Task.js'
```

¿Qué hace?

- `express`: Framework para crear el servidor web
- `mongoose`: Librería para conectar y trabajar con MongoDB
- `cors`: Permite que tu React (puerto 5173) se comunique con el servidor (puerto 5000)
- `dotenv`: Lee variables de entorno desde archivo `.env`
- `Task`: Tu modelo de datos (esquema de las tareas)

2. **Configuración Inicial**

```
dotenv.config()
const app = express()
const PORT = process.env.PORT || 5000
```

¿Qué hace?

- Carga las variables del archivo `.env` (como MONGO_URI)
- Crea la aplicación Express
- Define el puerto (5000 en desarrollo, variable en producción)

3. **Middleware**

```
app.use(cors())
app.use(express.json())
```

¿Qué hace?

- `cors()`: Permite peticiones desde otros dominios (React → Express)
- `express.json()`: Convierte el JSON que envía React en objetos JavaScript

4. **Conexión a MongoDB**

```
mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log('✅ Conectado a MongoDB Atlas'))
  .catch(err => console.error('❌ Error al conectar a MongoDB:', err))
```

¿Qué hace?

- Se conecta a tu base de datos MongoDB Atlas
- Usa la URL de conexión desde tu archivo `.env`
- Muestra mensajes de éxito o error

5. **Rutas de la API**

Ruta de prueba:

```
app.get('/', (req, res) => {
    res.send('API ToDo funcionando 🚀')
})
```

- **URL**: `GET /`
- **Función**: Confirma que el servidor funciona

Obtener todas las tareas:

```
app.get('/tasks', async (req, res) => {
  const tasks = await Task.find()
  res.json(tasks)
})
```

- **URL**: `GET /tasks`
- **Función**: Busca todas las tareas en MongoDB y las devuelve como JSON

Crear nueva tarea:

```
app.post('/tasks', async (req, res) => {
  const task = new Task({ text: req.body.text, completed: false })
  await task.save()
  res.status(201).json(task)
})
```

- **URL**: `POST /tasks`
- **Función**: Recibe texto desde React, crea nueva tarea, la guarda en MongoDB

Eliminar tarea:

```
app.delete('/tasks/:id', async (req, res) => {
  await Task.findByIdAndDelete(req.params.id)
  res.sendStatus(204)
})
```

- **URL**: `DELETE /tasks/:id`
- **Función**: Elimina la tarea con el ID especificado

6. **Inicio del Servidor**

```
app.listen(PORT, () => {
  console.log(`🚀 Servidor escuchando en http://localhost:${PORT}`)
})
```

¿Qué hace?

- Pone el servidor a "escuchar" en el puerto 5000
- Muestra mensaje confirmando que está funcionando

🔄 Flujo Cliente-Servidor

Al cargar la app (useEffect):

```
React (puerto 5173) → GET /tasks → Express → MongoDB → devuelve tareas → React
muestra lista
```

Al agregar tarea:

```
React → POST /tasks + {text: "nueva tarea"} → Express → MongoDB (guarda) →
devuelve tarea creada → React actualiza estado
```

Al eliminar tarea:

```
React → DELETE /tasks/123abc → Express → MongoDB (elimina) → React quita de la
lista
```

📁 Estructura de Archivos

En tu servidor necesitas:

- `.env` con `MONGO_URI=tu_url_de_mongodb`

- `models/Task.js` con el esquema de Mongoose
- `package.json` con las dependencias

Dependencias principales:

```
{
  "dependencies": {
    "express": "^4.x.x",
    "mongoose": "^7.x.x",
    "cors": "^2.x.x",
    "dotenv": "^16.x.x"
  }
}
```

🛠️ Tecnologías Utilizadas

Tecnología	Propósito
Express.js	Framework web para Node.js
MongoDB	Base de datos NoSQL
Mongoose	ODM para MongoDB
CORS	Manejo de peticiones cross-origin
React	Frontend/Cliente
Vite	Herramienta de desarrollo

📝 Notas Importantes

1. **Puertos**: El servidor corre en puerto 5000, React en 5173
2. **CORS**: Necesario para comunicación entre diferentes puertos
3. **Async/Await**: Se usa para operaciones asíncronas con MongoDB
4. **REST API**: Siguiendo convenciones RESTful para las rutas
5. **Error Handling**: Básico, se puede mejorar con try/catch

🚀 Próximos Pasos

- Agregar validación de datos
- Implementar manejo de errores más robusto
- Agregar funcionalidad de completar tareas
- Implementar autenticación de usuarios
- Agregar tests unitarios