Trabajo Práctico Integrador.

Ciclo Lectivo 2025.

Arquitectura Web - Modalidad Online.

Universidad de Palermo.

Autora: Magalí Sarmiento.

Docentes: Diego Marafetti y Adrián Ezequiel Martínez.

VISIÓN GENERAL

El presente trabajo práctico consiste en desarrollar una aplicación web para el Manejo de Stock del emprendimiento Deli Lunch. La app web permite mostrar todos los productos que requiere mi mamá para cocinar, tanto su disponibilidad como su agotamiento, Asimismo, permite que ella modifique todo el stock, pudiendo adicionar, eliminar y modificar productos.

FUNCIONALIDADES DE LA APLICACIÓN WEB.

1. **FrontEnd**: encargado de la presentación solamente. Se utiliza el framework React, HTML y CSS.

    1.1. Se debe implementar al menos 1 CRUD o ABM.

    1.2. Se debe implementar al menos 1 pantalla de reporte.

    1.3. La implementación de login es completamente opcional.

2. **BackEnd**: implementa la lógica de negocio y la persistencia de datos.

    2.1. Debe exponer una RESTful API (nivel 2 mínimo) para consumir los distintos servicios. Acá se evalúa el diseño y las buenas prácticas.

    2.2. Se utiliza un motor de base de datos para la persistencia. Cuando la aplicación se levanta, se carga un set de datos por default para poder testear correctamente.

3. **Testing.**

    3.1. Realizar testing sobre los endpoints. Pueden ser utilizando alguna herramienta para tal fin, o bien codificados. No realizar unit testing en el frontend.

4. **Build y Ejecución.**

    4.1. Incluir un readme con los pasos para hacer un build y ejecutar el proyecto. Incluir algún script o hacer uso de alguna herramienta (npm por ejemplo).

ÍNDICE.

Descripción de la aplicación desarrollada.

Propuesta de Valor de la Aplicación.

“**Deli Lunch**” es una aplicación web pensada para optimizar el manejo de stock en emprendimientos familiares, especialmente en aquellos donde no se cuenta con conocimientos técnicos o sistemas de gestión complejos. La idea surgió a partir de observar cómo mi mamá, quien cocina y vende comidas caseras, llevaba el control de su inventario utilizando métodos manuales como anotaciones en papel y planillas.

Con el tiempo, noté que este sistema le generaba confusión, pérdida de información y una carga administrativa innecesaria. Fue entonces cuando decidí diseñar una herramienta digital simple, pero poderosa, que le permitiera visualizar y administrar el stock de sus productos de forma clara y sin complicaciones.

La propuesta de valor de Deli Lunch radica en su **accesibilidad y facilidad de uso**, permitiendo que cualquier persona, incluso sin experiencia técnica, pueda:

- Ver rápidamente qué productos están disponibles o agotados.
- Agregar, modificar o eliminar productos del stock con unos pocos clics.
- Tener un mejor control del negocio y reducir errores humanos.
- Ver un reporte con los ingresos mensuales.
- Agendar los pedidos del negocio.

En resumen, Deli Lunch transforma una gestión rudimentaria y propensa a errores en una solución moderna, accesible y útil para el día a día, no solo ayudando a mi mamá, sino también a cualquier emprendedor que quiera organizar mejor su producción sin depender de herramientas complejas o costosas.

Motivación personal

La aplicación nace de la necesidad concreta de mi mamá, quien administra su emprendimiento de comidas caseras con métodos tradicionales (anotaciones en papel, planillas manuales). Fue a partir de observar sus dificultades que surgió la idea de diseñar una herramienta digital **accesible, simple y efectiva**, pensada para ayudarla a mejorar la gestión de su producción diaria sin requerir conocimientos técnicos.

Funcionalidades destacadas del sistema.

CRUD de productos.

Permite crear, leer, actualizar y eliminar productos del stock. Cada comida tiene asociada una cantidad disponible, nombre y descripción.

Alertas visuales por stock bajo

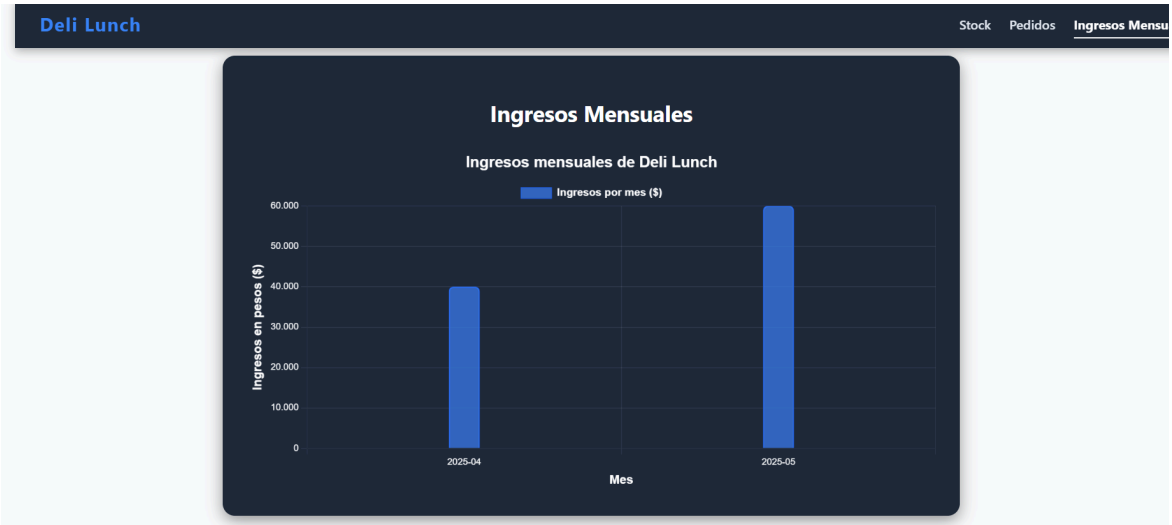
Para facilitar una toma de decisiones más rápida, el sistema resalta automáticamente en **color rojo** aquellos productos cuyo stock es menor a 5 unidades. Esta funcionalidad busca **evitar que se agoten productos sin que el usuario lo note**, y anticiparse a los faltantes. Por ejemplo: si un producto tiene 4 unidades restantes, aparecerá destacado en rojo en la lista principal.

Gestión de Pedidos

La funcionalidad actúa como una agenda digital de encargos y ayuda a reducir olvidos o confusiones. El usuario puede registrar **quién hizo un pedido**, **para qué fecha debe estar listo**, **el tipo de envío** (retiro en domicilio o entrega a domicilio), **el estado del pedido** y **el precio** que se le cobra al cliente. Esto le permite llevar un registro más claro de los compromisos asumidos y organizar mejor su producción y logística.

Pantalla de reporte de ingresos mensuales.

Se incorporó una nueva funcionalidad que permite visualizar los ingresos generados por la venta de productos, agrupados por mes. Desde el frontend, se creó una pantalla de reportes donde se consume esta información y se presenta de forma gráfica mediante un gráfico de barras, facilitando a la usuaria interpretar la evolución de sus ingresos mes a mes, sin necesidad de manejar herramientas externas o bases de datos. La funcionalidad fue pensada especialmente para emprendedores que buscan un control simple y accesible de sus ventas a lo largo del tiempo.



La arquitectura.

Arquitectura en Capas (Layered Architecture)

Para el desarrollo de la aplicación Deli Lunch, se optó por implementar una Arquitectura en Capas (Layered Architecture) combinada con el patrón MVC en el backend. La decisión busca separar claramente la interfaz, la lógica de negocio y la comunicación con el backend, mejorando la organización del código, facilitando el mantenimiento y permitiendo trabajar en paralelo en

frontend y backend. Esto brinda flexibilidad, facilita las pruebas y permite escalar o modificar partes del sistema sin afectar al resto.

Patrón de diseño MVC: Justificación y Ejemplo.

En el desarrollo del backend de la aplicación Deli Lunch, se implementa una arquitectura basada en el patrón Modelo-Vista-Controlador (MVC), adaptada para construir una API RESTful. Esta elección responde a la necesidad de organizar el código de forma clara y eficiente, facilitando la escalabilidad y el mantenimiento del proyecto, permitiendo que cada componente evolucione de forma más independiente.

La estructura de carpetas refleja la separación lógica en tres capas principales:

- **Modelos (/db/models/):** se definen las entidades principales de la aplicación, como las comidas (los productos) y los pedidos. Estos modelos representan la estructura y reglas de negocio de los datos, y se encargan de interactuar directamente con la base de datos SQLite, garantizando la integridad y coherencia de la información.

- **Controladores (/controllers/):** los controladores, como comidaController.js y pedidosController.js, actúan como intermediarios entre los modelos y las rutas. Ellos reciben las solicitudes del cliente, ejecutan la lógica necesaria, consultan o modifican los datos a través de los modelos, y finalmente devuelven las respuestas adecuadas.

- **Rutas (/db/routes/):** Las rutas definen los puntos de acceso de la API (endpoints) y asignan cada solicitud HTTP a su correspondiente controlador. Por ejemplo, la ruta /api/comidas direcciona a los métodos de comidaController.

En pocas palabras, el backend se encarga del **Modelo**, **Controlador** y **Rutas**, y el el frontend (React) es quien se encarga de la **vista**, mostrando la información al usuario y gestionando la interacción, la interfaz visual.

La arquitectura del backend de Deli Lunch puede compararse con el funcionamiento de un restaurante. La vista (frontend en React) sería el mozo, que toma el pedido del cliente y se lo pasa a la cocina. El controlador actúa como el jefe de cocina, que recibe el pedido del mozo, decide qué hacer y da las instrucciones necesarias. Los modelos representan la despensa y la cocina, donde están guardados los ingredientes y recetas, es decir, los datos que se consultan o modifican en la base de datos (SQLite). Todo este proceso está organizado mediante rutas, que serían los caminos definidos para que los pedidos lleguen correctamente desde el cliente hasta la cocina y regresen con la comida lista. Esta estructura permite que cada parte del sistema cumpla una función clara, facilitando el mantenimiento y crecimiento de la aplicación.

Lenguaje de programación.
JavaScript.

Java se eligió por su portabilidad, ya que puede ejecutarse en cualquier sistema gracias a la JVM. Es un lenguaje orientado a objetos, lo que facilita la organización y el mantenimiento del código. Además, ofrece seguridad, manejo automático de memoria y soporte para múltiples hilos, ideal para aplicaciones interactivas. Finalmente, su estabilidad, flexibilidad, y capacidad para crear aplicaciones completas mediante tecnologías como Node.js, Express y frameworks del lado del cliente, lo convierten en una opción confiable para el desarrollo de Deli Lunch.

Estructura del Frontend - DeliLunch

React.

El frontend de DeliLunch está construido con React, y se encarga exclusivamente de la presentación y la interacción con el usuario, consumiendo una API REST desarrollada en el backend. A continuación se describe cada parte del proyecto:

Estructura del frontend.

Frontend

├── package-lock.json

├── package.json

├── .env

├── .gitignore

├── **/public**

│   ├── index.html

│   ├── manifest.json

│   ├── robots.txt

├── **/src**

│   ├── index.js

│   ├── index.css

│   ├── App.js

│   ├── **/components**

│   │   ├── FormularioComida.js

│   │   ├── FormularioPedidos.js

```
│   │   ├── ListaDeComidas.js
│   │   ├── ListaDePedidos.js
│   │   ├── Navbar.js
│   │   └── Navbar.css
│   ├── /pages
│   │   ├── Home.js
│   │   ├── Pedidos.js
│   │   └── ReporteIngresos.js
│   ├── /services
│   │   ├── comidaService.js
│   │   └── pedidoService.js
│   └── /styles
│       └── App.css
```

Descripción básica de la estructura principal del frontend.

/src/ carpeta principal con el código fuente.

- **/components** Componentes reutilizables de la UI:
    - FormularioComida.js: formulario para agregar comidas.
    - FormularioPedidos.js: formulario para crear pedidos.
    - ListaDeComidas.js: tabla con comidas y opciones para editar/eliminar.
    - ListaDePedidos.js: lista de pedidos realizados.
    - Navbar.js y Navbar.css: barra de navegación y estilos.
- **/pages** Páginas principales:
    - Home.js: inicio con formulario y lista de comidas.
    - Pedidos.js: página para registrar y ver pedidos.
    - ReporteIngresos.js
- **/services** Funciones para comunicación con backend:

    - comidaService.js: operaciones HTTP para comidas.
    - pedidoService.js: operaciones HTTP para pedidos.
- **/styles** Estilos globales CSS.
    - App.css.

/public Archivos estáticos base (HTML, manifest, robots.txt).

Otros archivos.

- App.js: componente principal con rutas.
- index.js: punto de entrada de React.
- index.css: estilos base.
- package.json: dependencias y scripts.
- .env: variables de entorno.
- .gitignore: archivos ignorados por Git.
- package-lock.json: versiones exactas de dependencias.

CRUD implementado.

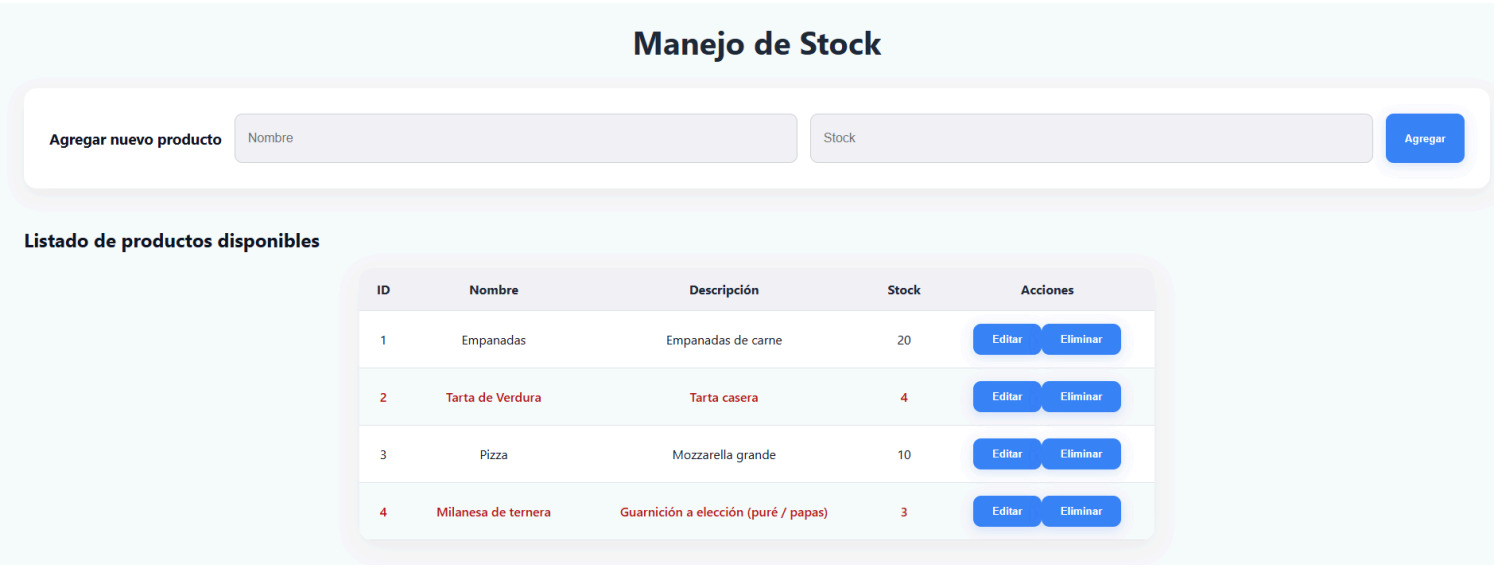
Manejo de stock de DeliLunch.

En el sistema Deli Lunch, se implementó un CRUD completo sobre las comidas disponibles en stock.

Las operaciones que se pueden realizar sobre las comidas disponibles son:

- **Crear:** A través del formulario de carga (FormularioComida.js), se pueden agregar nuevas comidas indicando nombre y cantidad en stock.
- **Leer:** Las comidas registradas se muestran en una tabla (ListaDeComidas.js), donde se listan el ID, nombre y stock disponible.
- **Actualizar:** Desde la misma lista, el usuario puede seleccionar una comida y modificar su nombre o stock, actualizándose los datos mediante un formulario editable.
- **Eliminar:** En cada fila de la tabla se incluye un botón para eliminar esa comida del sistema.

Este CRUD es accesible desde la **pantalla principal (Home.js)** y permite una gestión sencilla y completa del stock de comidas.



Manejo de pedidos.

Las operaciones que se pueden realizar sobre los pedidos son:

- **Crear:** a través del formulario de carga (FormularioPedidos.js), se pueden agregar nuevos pedidos indicando; el cliente, la fecha, el tipo de envío, el estado y el precio.
- **Actualizar:** Desde la misma tabla, se puede actualizar el estado del pedido para marcarlo como terminarlo.
- **Eliminar**: En cada fila de la tabla se incluye un botón para eliminar el pedido en caso de equivocación o cancelación del mismo.

Este CRUD es accesible desde la **pantalla principal (Pedidos.js)** y permite una gestión sencilla y completa de los pedidos.

Sobre la decisión de no implementar login

En esta primera versión del sistema, se decidió **no implementar un sistema de login**. Esta decisión se justifica en que el sistema será utilizado por **una única usuaria**, quien es responsable del manejo del stock y los pedidos. Al no haber múltiples usuarios ni necesidad de restricción de acceso, se consideró que una autenticación resultaría innecesaria en esta instancia, priorizando así la simplicidad y facilidad de uso.

Estructura del BackEnd - DeliLunch

Node.js con Express.js

Se selecciona Node.js con Express para el backend por su compatibilidad con JavaScript del frontend, su bajo nivel de complejidad para proyectos simples y su excelente soporte para construir APIs RESTful de forma rápida y ordenada.

RESTful API (nivel 2 mínimo)

La aplicación DeliLunch implementa una API RESTful que alcanza el Nivel 2 del modelo de madurez de Richardson. Este nivel se caracteriza por el uso correcto de múltiples recursos bien definidos mediante URLs y la utilización adecuada de los métodos HTTP estándar (GET, POST, PUT, DELETE) para operar sobre ellos. En esta API, los recursos como /api/comidas y /api/pedidos representan entidades del dominio, y cada verbo HTTP cumple una función específica: GET para obtener información, POST para crear nuevos registros, PUT para actualizarlos y DELETE para eliminarlos. La estructura uniforme de las rutas mejora la claridad, la escalabilidad y la mantenibilidad del sistema.

Estructura del Backend.

Backend

├── package-lock.json

├── package.json

├── initDB.js

├── server.js

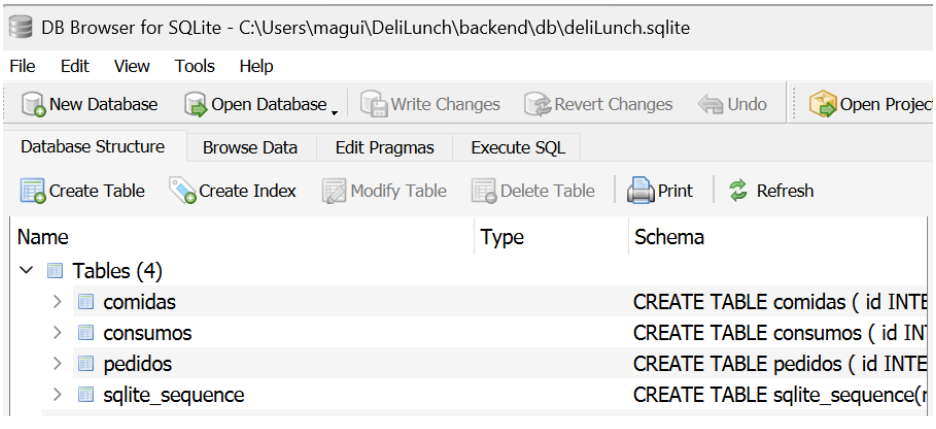
├── **/controllers.**

```
│   ├── comidaController.js
│   ├── pedidoController.js
├── /db
│   ├── db.js
│   ├── deliLunch.sqlite
│   ├── seed.js
│   ├── /models
│   │   ├── comida.js
│   │   ├── pedido.js
│   ├── /routes
│   │   ├── comida.js
│   │   └── pedidos.js
│   ├── /tests
│   │   ├── comida.test.js
│   │   ├── pedido.test.js
```

Base de datos.

Base de datos SQLite.

Para este trabajo práctico se eligió SQLite como base de datos por su simplicidad y facilidad de uso. Al ser una base embebida, no requiere instalar ni configurar un servidor externo, lo cual agiliza el desarrollo. Además, permite trabajar con persistencia de datos en un único archivo, ideal para proyectos pequeños como DeliLunch. Su portabilidad, compatibilidad con Node.js y bajo mantenimiento la hacen perfecta para prácticas académicas donde el foco está en aplicar conceptos de arquitectura web, APIs REST y CRUD.



Otras justificaciones.

Justificación sobre la ausencia de validaciones.

En esta entrega del proyecto *DeliLunch*, se prioriza la implementación funcional del sistema de gestión de stock y pedidos, enfocándose en lograr un flujo completo de operaciones CRUD entre el frontend y el backend. Las validaciones de datos en el formulario fueron omitidas intencionalmente, ya que el sistema será operado inicialmente por una única usuaria (mi madre), en un entorno controlado y de confianza. Esto permite concentrar los esfuerzos en aspectos estructurales y de integración del sistema. No obstante, se contempla incorporar validaciones en futuras versiones para reforzar la robustez y prevenir errores de carga en escenarios de uso más amplios o menos controlados.

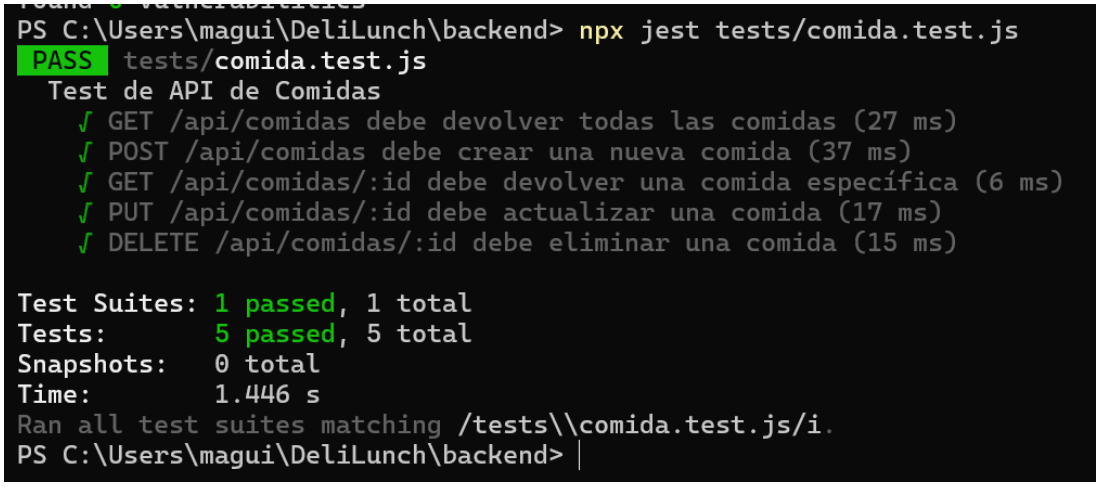
Testing.

Testing sobre los endpoints.

Para garantizar el correcto funcionamiento de la API, se implementaron pruebas automatizadas sobre los endpoints utilizando Jest como framework de testing junto con Supertest para simular peticiones HTTP. Estas pruebas permiten verificar que cada operación de la API responde correctamente ante diferentes escenarios.

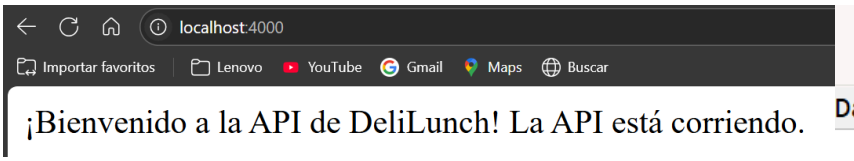
En el backend se desarrollaron dos archivos de pruebas: comida.test.js y pedido.test.js, ubicados dentro de la carpeta tests. En ellos se testean los principales endpoints CRUD de comidas y pedidos (GET, POST, PUT, DELETE), así como también un endpoint especial de reporte para los ingresos mensuales.

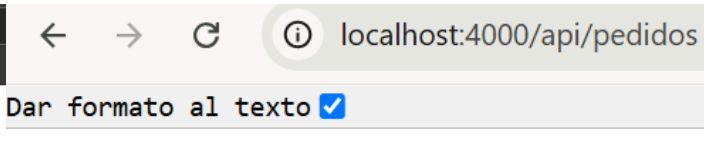
Cada test evalúa tanto el código de estado HTTP como el contenido de la respuesta, validando que los datos sean coherentes y que el servidor actúe conforme a lo esperado. Las pruebas se ejecutan con el comando *npx jest*, el cual está configurado en el package.json mediante el script "test": "jest". Esta estrategia de testing ayuda a detectar errores temprano y aporta confiabilidad al desarrollo de la API.

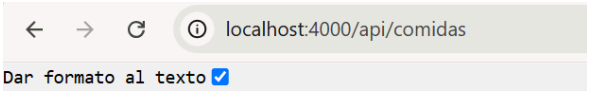


```
PS C:\Users\magui\DeliLunch\backend> npx jest tests/pedido.test.js
  console.log
    Datos recibidos para crear pedido: {
      cliente: 'Juan Perez',
      fecha_entrega: '2025-05-30',
      tipo_envio: 'Retira',
      estado: 'Pendiente',
      precio: 2500
    }

      at log (controllers/pedidoController.js:12:11)

 PASS  tests/pedido.test.js
  Test de API de Pedidos
    √ POST /api/pedidos debe crear un nuevo pedido (74 ms)
    √ GET /api/pedidos debe devolver todos los pedidos (8 ms)
    √ GET /api/pedidos/ingresos-mensuales debe devolver reporte de ingresos (5 ms)
    √ DELETE /api/pedidos/:id debe eliminar un pedido (14 ms)
    √ PUT /api/pedidos/:id debe actualizar un pedido (6 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.974 s, estimated 1 s
Ran all test suites matching /tests\\pedido.test.js/i.
PS C:\Users\magui\DeliLunch\backend>
```

Pruebas de funcionamiento personales.








```
[
  {
    "id": 1,
    "nombre": "Milanesa de ternera",
    "descripcion": "Milanesa con queso",
    "stock": 25
  },
  {
    "id": 3,
    "nombre": "Pizza",
    "descripcion": "Mozzarella grande",
    "stock": 10
  },
  {
    "id": 4,
    "nombre": "Milanesa de ternera",
    "descripcion": "Guarnición a elección (puré / papas)",
    "stock": 3
  },
  {
    "id": 5,
    "nombre": "Milanesa",
    "descripcion": "Milanesa de pollo",
    "stock": 30
  },
  {
    "id": 6,
    "nombre": "Papas",
    "descripcion": null,
    "stock": 2
  }
]
```

```
[
  {
    "id": 4,
    "cliente": "Joaquín",
    "fecha_entrega": "2025-04-23",
    "tipo_envio": "Retira",
    "estado": "Pendiente",
    "precio": 40000
  },
  {
    "id": 5,
    "cliente": "Milagros Longarela ",
    "fecha_entrega": "2025-05-23",
    "tipo_envio": "Entrega",
    "estado": "Pendiente",
    "precio": 60000
  },
  {
    "id": 11,
    "cliente": "Magali",
    "fecha_entrega": "2025-06-01",
    "tipo_envio": "Entrega",
    "estado": "Pendiente",
    "precio": 12000
  }
]
```

```
PS C:\Users\magui> cd DeliLunch
PS C:\Users\magui\DeliLunch> cd backend
PS C:\Users\magui\DeliLunch\backend> Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
PS C:\Users\magui\DeliLunch\backend> npm start

> backend@1.0.0 start
> node server.js

Servidor corriendo en http://localhost:4000
Ruta / llamada
Datos recibidos para crear pedido: {
  cliente: 'Magali',
  fecha_entrega: '2025-06-01',
  tipo_envio: 'Entrega',
  estado: 'Pendiente',
  precio: 12000
}
Ruta / llamada
Ruta / llamada
Ruta / llamada
```

```
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```