Trabajo Práctico 2

Teoría + Mini-proyectos (TypeScript/Express/Zod + Jest o Vitest)

Objetivo general. Practicar TDD (Rojo → Verde → Refactor) construyendo una API pequeña con TypeScript + Express, validaciones con Zod y tests **unitarios** + **integración** (Jest **o** Vitest + Supertest). Las **preguntas teóricas** son comunes a todos; cada grupo desarrolla **un mini-proyecto distinto** de la lista.

1) Requisitos técnicos

- Node 18+ y npm.
- TypeScript, Express, Zod.
- Testing: **Jest** (ts-jest) **o Vitest** + **Supertest**.
- Validación en bordes con Zod (body/query/params). Reglas de negocio en servicios (probadas con unit tests).
- Separar **app** de **server**: `makeApp()` sin `listen` en tests.

2) Modalidad y organización

- **Evidencia de TDD:** Describan como hicieron `test rojo → implementación mínima → refactor`. (en el readme de github)
- **Repositorio público o privado compartido con docentes**.

3) Entregables

1. **Código** de la API con tests unitarios + integración en verde.
2. **Historial de commits** mostrando el ciclo TDD por historia (user story).
3. **Cobertura**: ≥ **80%** en archivos modificados.
4. **Matriz de casos** (tabla CA ↔ tests) en README o en el PR.
5. **Guía de ejecución**: scripts npm, requisitos, comandos para correr tests.

4) Preguntas de teoría (comunes para todos)

Responda en un archivo `TEORIA.md` (máx. 1–2 párrafos por ítem, salvo que se pida código):

1. Explique el ciclo **Rojo** → **Verde** → **Refactor** y por qué es importante el tamaño de los pasos.
2. Diferencie **tests unitarios**, **de integración** y **E2E** en APIs.
3. ¿Qué es un **doble de prueba**? Defina **mock**, **stub** y **spy** y cuándo conviene cada uno.

4. ¿Por qué es útil separar `app` de `server`? Muestre (en 8–10 líneas) un ejemplo mínimo con `makeApp()` y un test de integración con Supertest.
5. Zod: diferencia entre `parse` y `safeParse`. ¿Dónde usaría cada uno en una ruta Express y por qué?
6. Dé **dos ejemplos** de **reglas de dominio** que deben probarse con tests unitarios (no sólo validación de entrada).
7. ¿Qué **malos olores** suele haber en suites de tests? Dé 3 ejemplos (naming, duplicación, asserts débiles, mocks frágiles, etc.).
8. ¿Cómo trazará **criterios de aceptación** ↔ **tests**? Incluya un mini ejemplo de tabla con 2 filas.
9. ¿Por qué no perseguir 100% de cobertura a toda costa? Mencione riesgos/limitaciones.
10. Defina y dé un ejemplo de **helper**/builder para tests.

5) Mini-proyecto

Para este proyecto implementar **al menos**: 3–5 endpoints, validaciones con Zod, **tests unitarios** de reglas de negocio y **tests de integración** del contrato HTTP (status/body/errores). Incluya ejemplos `curl` en README.

Pedidos de Pizzería

- **Endpoints:**

  `POST /orders`,

  `GET /order/:id`,

  `POST /orders/:id/cancel`,

  `GET /orders?status`.

- **Reglas:** `size ∈ {S,M,L}`, máx. 5 toppings; **precio calculado** (size + toppings); no se puede cancelar si `status = delivered`.
- **Validaciones:** body con `items[]` no vacío; `address` min 10.
- **Errores:** 422 si `items` vacío; 409 si intenta cancelar entregado.

6) Reglas de TDD para este TP

1. Escribir primero el **test** del caso (unitario) → **rojo**.
2. Implementar **lo mínimo** en el servicio → **verde**.
3. Agregar test **de integración** del endpoint (status, body, errores) → **rojo** → **verde**.
4. **Refactor** (nombres, helpers, duplicación) manteniendo tests en verde.
5. Repetir por cada historia (user story).

6.

7) Formato de entrega

- Link al **repositorio**, README con: scripts, instrucciones de ejecución, ejemplos `curl` y **Matriz de casos**.
- Indicar si usan **Jest** o **Vitest** y cómo correr `coverage`.
- Marcar en el README las **user stories** abordadas.

8) Plantilla de matriz de casos (copiar/pegar)

ID	Caso / Descripción	Precondición	Input	Acción	Resultado esperado	Test
CA1						
CA2						
ER R1						

9) Reglas administrativas

- No subir secretos/credenciales.
- Si usan datos persistentes, usar **in-memory** o un stub/adapter para la DB (no se requiere DB real).
- Cada integración con librerías externas debe aislarse para poder testear.