Phase 1. Initialisation et modèle de données

- Créer un nouveau projet Rust, ajouter Tokio comme dépendance.
- Définir des types ci-dessous :
    - enums : Burger, Snack, Drink, Size, ItemKind
    - structs : OrderLine (item, quantité, taille optionnelle), Order (id, lines), PreparedItem (order_id, item)

Phase 2. Génération aléatoire des commandes

- Ajouter un générateur simple de commandes.
    - Nombre de commandes : N (ex. 20)
    - Par commande : 2 à 6 lignes, quantités 1 à 2
    - Choix aléatoire des items (menus) et tailles pour les boissons
- Fixer une graine reproduire les mêmes tests.

Phase 3. Concurrence de base, postes de préparation

- Mettre en place 3 files de messages (mpsc) : grill, friteuse, boissons.
- Définir un Job par item (order_id, item_kind, canal oneshot pour confirmer la fin).
- Lancer 1 à 2 workers par poste qui reçoivent un Job, attendent une durée pour simuler la préparation, renvoient “terminé”.

Phase 4. Traitement d’une commande, parallélisme intra-commande

- Écrire process_order(order, kitchen) :
    - découper la commande en Jobs, router chaque Job vers le bon poste,
    - attendre la fin de tous les items (oneshot), puis marquer la commande “terminée”.
- Exécuter plusieurs commandes en parallèle pour montrer multi-commande + multi-poste.

Phase 5. Sortie démonstrative et mesure simple

- Ajouter des logs courts : début/fin de commande, début/fin d’item.
- Mesurer le temps par commande et afficher une synthèse.

Phase 6. Point de démonstration

- Expérience simple : changer le nombre de workers (ex. friteuse 1 vs 2) et relancer la même série de commandes.
- Comparer le temps total et quelques temps de commande pour illustrer l’impact de la concurrence.

Phase 7. Finition

- Séparer le code en 2 à 3 fichiers (model, kitchen, main), ajouter un README minimal (comment lancer, quels primitives : spawn, mpsc, oneshot, comment modifier le nombre de workers).