I began by sketching the core loop on paper to clarify scope, roles, and dependencies. At the beginning it was single player, but in the mean time i was evolving it to Multiplayer. From there I implemented the systems primarily in C++: a generic interaction component, data-driven `APickupActor` with replicated state and editor-configurable meshes per `EIngredientState`, and a `CookwareStation` that handles three workflows: CuttingBoard (Fish Raw→Sliced), Stove/Pot (Rice Raw→Cooked), and MixingTable (combines Sliced Fish, Cooked Rice, and Seaweed to produce a Sushi “final dish”). I added `PlateActor` logic to accept processed items or a final dish, and a `DropZone` that hands items to stations and preserves the instigating pawn so the character can expose a replicated `bIsProcessingAtStation` flag for the AnimBP. To support a flexible economy of ingredients, I created `IngredientSpawner` (plus Fish/Rice/Seaweed variants) that spawn pickups via Interact inside a trigger volume.

On the UX side, I built a Main Menu level with UMG widgets for Host, Join, and Quit, wired to level open commands (`?listen`/IP). I profiled packaging settings (target Windows), defined startup/cooked maps, and ensured input mappings and collisions were consistent.

I’m pleased with the code quality, replication hygiene, and data-driven meshes, especially under time constraints and third-party setbacks. I’m also disappointed I couldn’t deliver the fully complete multiplayer loop and polished UI/score flow. Still, I pushed to cover the core gameplay path end-to-end and leave clear extension points for orders, scoring, and session management.