la logica dell'app è questa. Anselmo e Monica si siedono al tavolo del ristorante. Arriva il cameriere che dal suo cellulare, attraverso l'app genera un qr code univoco temporizzato (che scade appena i clienti pagano il conto). Anselmo non ha l'app installata sul suo cellulare mentre Monica si. Il cameriere genera un QR che contiene un codice/sessione temporanea e scrive una sessione su Firestore; Monica apre l’app direttamente e una volta aperta scannerizza il qr code, mentre Anselmo (senza app) scarica prima l’app dallo store e poi scansiona il codice dall’interno dell’app per unirsi alla sessione. Nessun deferred deep link, nessuna apertura automatica dallo store. L'app è ora aperta su entrambi i dispositivi e riconosce in automatico la lingua di default. dopo il caricamento della splash page l'app riconosce in automatico il numero del tavolo permettendo ad Anselmo e Monica di ordinare ognuno per sé direttamente dal proprio dispositivo. L’app segue due flussi, serve cioè sia al cliente che allo staff (proprietario, cassiere, cuoco, cameriere). Il cliente avrà sempre a disposizione il menu di navigazione basso con i link alla pagina Home, Offerte, Ordina, I miei Punti e altro. Cliccando altro, si aprirà una nuova pagina con altri link uno dei quali sarà I miei ordini (uno storico degli ordini effettuati) e un altro sarà Area Staff. Appena Anselmo apre l’app clicca la sezione Offerte per vedere le offerte ed ordinarne eventualmente. Poi apre la sezione Ordina dove appare il menu. Ordina le sue pietanze ed invia l’ordine. Subito dopo aver premuto il pulsante invia ordine l’app gli chiede se intende usufruire ed accumulare i punti messi a disposizione dal proprietario (esempio 1 punto ogni 1 euro di spesa). Se Anselmo risponde no allora l’ordine arriva direttamente in cucina senza ulteriori ritardi, altrimenti se risponde di si l’app gli chiede di registrarsi, chiedendo il numero di cellulare ed una password (recuperabile tramite mail attraverso un servizio gratuito). Una volta registrato all’utente Anselmo verranno accreditati i punti a seconda delle pietanze scelte. Monica, che aveva già usato l’app ed accumulato dei punti, viene riconosciuta in automatico ed anche lei opta per prendere i punti che le vengono accreditati in automatico. L’app deve prevedere un sistema attraverso il quale un utente può regalare parte dei suoi punti ad un altro utente. Il fine dei punti è quello di sbloccare determinati livelli ed usufruire di sconti, bibite gratis, pasti gratis ecc. ecc. Gli ordini separati partiti dai cellulari di Anselmo e Monica arrivano sullo schermo della cucina raggruppati nello stesso tavolo. Il cuoco ha la possibilità di cliccare ogni singola pietanza ordinata per trasformare lo stato della pietanza stessa. Appena arriva in cucina la pietanza è allo stato in attesa, appena il cuoco comincia a prepararla cliccando il pulsante la farà diventare in preparazione. Appena è pronta il cuoco cliccando il pulsante la farà passare a pronta. Quando la pietanza singola (non l’ordine intero del tavolo) sarà dichiarata pronta deve scomparire dallo schermo della cucina ed apparire contemporaneamente in due luoghi diversi: nell’archivio della cucina (da dove può essere richiamata e rimessa in preparazione qualora il cuoco l’avesse dichiarata pronta per errore) e nello schermo della sala del cameriere. Il cameriere deve avere la possibilità di vedere in tempo reale apparire tutte le pietanze dichiarate pronte dal cuoco per poterle poi consegnare ai tavoli, inoltre deve poter visualizzare, senza poter agire, le pietanze della cucina che sono in preparazione. Ogni pietanza che arriva dalla cucina per essere consegnata sarà visualizzata sullo schermo del cameriere come “da consegnare”. Una volta dichiarata consegnata (cliccando il pulsante accanto alla pietanza stessa) quella pietanza scompare dallo schermo del cameriere ed appare nell’archivio del cameriere da dove può essere ripristinata in caso di errore. Il flusso dei clienti si ferma dopo l’ordine. Dopo aver ordinato le pietanze il compito del’app è terminato. Il flusso dello staff oltre al cuoco e al cameriere coinvolge anche il cassiere ed il proprietario. Per ora il cassiere ed il proprietario hanno le stesse capacità cioè devono poter fare e vedere tutto. Il proprietario deve poter inserire le offerte che saranno visualizzate dai clienti nella sezione apposita. Deve poter creare il menu che sarà poi visualizzato nella schermata apposita. Deve poter vedere ed agire sull’archivio della cucina e su quello della sala, inoltre ha la possibilità di eliminare tutti gli ordini giornalieri per liberare il database e tenerlo aggiornato per il giorno dopo. Deve poter visualizzare le statistiche e richiamare un ordine di un tavolo specifico per fare il conto e stampare lo scontrino fiscale.

Perfetto — ecco un promemoria completo, operativo e copiabile che copre TUTTA la logica dell’app così come me l’hai descritta, dall’architettura ai singoli passi di implementazione / test / deploy. Ogni punto indica file target, criteri di successo misurabili e rischi; alla fine trovi una checklist sintetica per i test manuali (scenario Anselmo+Monica+staff).

**1) Contratto dell’app (inputs / outputs / error modes)**

* Input:
  + QR code / deep link contenente sessionId / codiceSessione
  + Azioni utente (aggiungi al carrello, invia ordine, segna pietanza pronta/servita, pagamento)
  + Dati amministratore (crea menu, crea offerte)
* Output:
  + SessionProvider aggiornato con sessioneCorrente (numero tavolo)
  + Ordini in Firestore con campo numeroTavolo
  + Stream real-time per staff (cucina/sala) e clienti
* Error modes:
  + Sessione non trovata / scaduta
  + Firestore non raggiungibile (emulator o rete)
  + Mismatch dati legacy (stringhe vs enum)
  + Permessi Firestore non corretti
* Success criteria:
  + Multi-device: due device connessi al medesimo sessionId vedono ordini raggruppati per tavolo e flussi staff funzionano.
  + Pagamento invalida sessione e client ricevono evento.

**2) Architettura di alto livello (componenti)**

* Frontend Flutter (provider + presentation widgets)
  + Providers principali: MenuProvider, CartProvider, OrdiniProvider, AuthProvider, SessionProvider
  + UI: Splash, Home, Menu, Offerte, Punti, Staff, Cucina, Sala, Archivio
* Backend lato app (Firestore)
  + Collections: pietanze, categorie, offerte, ordini, sessions, clienti, points
* Servizi core (lib/core/services)
  + MenuService, OrdiniService, FirebaseService, SessionService (nuovo), PointService, StampaService, QrGeneratorService
* Adapters / shims per compatibilità legacy: lib/adapters/\*

**3) Rischi principali da mitigare subito**

* Enum vs string: legacy usa stati testuali; core usa enum/index. Normalizzare la serializzazione e creare adapter di compatibilità.
* Race conditions all’avvio: Firebase init vs provider subscription (uso già di getter ritardati in MenuRepository, ma da verificare).
* Session persistence: sessione attiva solo in memoria => decidere se persistere (consiglio: server-side session doc in Firestore, client only pointer).
* Deferred deep link: non implementare (troppo fragile); usare scan in-app per install-first flow come deciso.
* Regole Firestore: proteggere la creazione di sessioni e scritture ordini.

**4) Roadmap step-by-step (ordinale e prioritario)**

Ogni step ha file target, criteri di successo e nota tecnica.

ALTA PRIORITÀ — requisito funzionale e stabilità

1. Schema sessione + Firestore rules (design)
   * File target: documento di progetto + firestore.rules (repo doc)
   * Schema sessions/{sessionId}: { sessionId (doc id), codice (string), numeroTavolo (int), idCameriere (string), createdAt (timestamp), expiresAt (timestamp), attiva (bool) }
   * Acceptance: schema definito e testabile su emulator.
   * Rischio: permessi di scrittura in produzione.
2. Implementare SessionService
   * File target: lib/core/services/session\_service.dart
   * API: createSession(...):Future<String>, joinSession(code):Future<SessionData>, endSession(sessionId):Future<void>, sessionStream(sessionId):Stream<SessionData>
   * Acceptance: tests unitari ed integration con Firestore emulator (create → read → stream → end).
   * Note: gestione retry/backoff; validate TTL.
3. Integrare QR generation (cammeriere)
   * File target: [qr\_tavolo\_dialog.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [qr\_generator\_service.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
   * Azione: cameriere chiama createSession → session doc creato → mostra QR (codice o sessionId).
   * UX: instruzioni chiare per utenti con e senza app.
4. Integrare scanner / joinSession (client)
   * File target: [selezione\_tavolo\_dialog.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (o scanner widget), SelezioneTavoloDialog, MenuScreen.\_checkout
   * Azione: scanner chiama joinSession → setta SessionProvider.setTavolo(numero) → salva optional sessionId in prefs.
   * Acceptance: dopo scan, SessionProvider ha sessione e UI mostra numero tavolo.
5. Invalidation al pagamento (hook cassiere/proprietario)
   * File target: flusso pagamento (es. [proprietario\_screen.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) o componente cassiere)
   * Azione: chiamare SessionService.endSession(sessionId); session doc aggiornata attiva=false
   * Acceptance: client riceve evento e SessionProvider.clearSessione()

MEDIA PRIORITÀ — coerenza dati, compatibilità, UX  
6) Estendere SessionProvider per ascolto remoto

* File target: [session\_provider.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
* Azione: se client ha sessionId salvato, sottoscrive sessionStream(sessionId) e chiude sessione al attiva=false.
* Acceptance: invalidazione remota propagata.

1. Normalizzare serializzazione enum ↔ legacy
   * File target: lib/core/models/\* ([pietanza\_model.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [ordine\_model.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)), adapters/\*
   * Azione: fromMap accetta sia int che string per stato; toMap standardizza su index; creare adapter helper per legacy.
   * Acceptance: test che legacy UI/DB funziona e nuovo code usa index.
2. Allineare menu/offerte persistence e fallback
   * File target: MenuService, MenuFirestoreService, [menu\_test\_data\_service.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
   * Azione: verificare timeout/fallback, coerenza attiva field (bool), evitare dual-type (string/bool) nel DB.
3. Clear plan per install-first flow (documentazione)
   * File target: README / QR dialog UI
   * Azione: mostrare messaggio chiaro su QR: “Se non hai l’app: scaricala e poi scorri e premi SCANSIONE QR”.

BASSA PRIORITÀ — osservabilità, refactor, hardening  
10) Logging e monitoring  
- File target: [logger.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), integrazione Crashlytics opzionale  
- Azione: log create/join/end sessions, ordini falliti, errori critici.

1. Tests automatici E2E con emulatori
   * File target: test/, pipeline CI
   * Azione: script che avvia Firestore emulator, esegue integration tests (create/join/end) e smoke tests UI.
2. Documentazione runbook & rollback
   * File target: tool/SESSION\_RUNBOOK.md, [README.md](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
   * Azione: passo-passo per ripristino usando [backups](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).

**5) File critici da controllare / modificare (mappa diretta)**

* Session / QR:
  + Nuovo: lib/core/services/session\_service.dart
  + Modifica: [qr\_tavolo\_dialog.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
  + Modifica: [selezione\_tavolo\_dialog.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (scanner)
  + Modifica opzionale: [splash\_screen.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (deep link inbound)
* Providers / models:
  + [session\_provider.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
  + [ordini\_service.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
  + [ordini\_provider.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
  + [pietanza\_model.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [ordine\_model.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (normalizzazione)
* UI (UX & text):
  + [offerte\_page.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
  + [menu\_screen.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (checkout path)
  + [staff\_screen.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [visualizza\_cucina\_screen.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), presentation/pages/sala/\*
* Other:
  + [qr\_generator\_service.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
  + [firebase\_service.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (use guest safe getters)
  + [database\_service.dart](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (legacy) — isolarlo o spostare in legacy/

**6) QA / Manual test checklist (scenario Anselmo + Monica + Staff)**

* Setup:
  + Avvia Firestore emulator (se disponibile).
* Steps:
  + Cam: genera QR (createSession): verifica doc sessions/{id} presente.
  + Monica (app installata): apre app → Scanner in-app → scansiona QR → joinSession e SessionProvider.setTavolo → UI mostra tavolo.
  + Monica: ordina e invia ordine → ordini in Firestore con numeroTavolo.
  + Anselmo (non ha app): scarica app, apre app, apre scanner, scansiona QR (se il QR è ancora valido) → joinSession ok; ordina → ordini appaiono nello stesso tavolo.
  + Cucina: verifica che ordini per stesso tavolo aggregati; cuoco cambia stato pietanza → passa a pronto → elemento scompare da cucina e appare in archivio cucina + appare nella schermata Sala/Cameriere come “da consegnare”.
  + Cameriere: segna consegnata → appare in archivio sala; può ripristinare.
  + Pagamento: cassiere clicca "conferma pagamento" → SessionService.endSession(sessionId) → session doc attiva=false → devices client vedono invalidazione e SessionProvider.clearSessione()
* Acceptance: tutti i passaggi funzionano e sono replicabili.

**7) Testing automatico (raccomandazioni)**

* Unit tests:
  + SessionService methods with Firestore emulator mocks.
  + model serialization tests (Pietanza/Ordine).
* Integration tests:
  + Firestore emulator scenario create→join→order→statusChange→endSession
* CI:
  + Step che avvia Firestore emulator, esegue test (timeout ragionevoli), report.

**8) Sicurezza & regole Firestore (sintesi)**

* Regole minime:
  + create session: only staff (claim or uid in staff list)
  + read session: allowed for all (così scanner in-app può leggere)
  + endSession: only staff or session owner
  + ordini collection: writes from authenticated clients allowed; verify numeroTavolo corrisponde a session attiva (optional)
* Acceptance: test con Firestore emulator e simulazione di utenti non autorizzati.

**9) Monitoraggio & osservabilità**

* Log critici: session create/join/end, order failures, stream disconnects.
* Metriche da esporre: session creations per ora, joins falliti, ordini per tavolo, latenza di stream.
* Crash reporting: Firebase Crashlytics (opzionale).
* Alert: join failures > X/min → notifica team.

**10) Criteri per richiedere un refactor esplicito**

Chiederò la tua approvazione PRIMA di refactorare se:

* File > ~250 LOC che fanno più di una responsabilità (es. SplashScreen: init + UI + navigation + fallback). Suggerimento: estrarre AppInitializer.
* MenuService, OrdiniProvider, o DatabaseService contengono mischiati persistence + business + presentation: propongo estrazione in repository / use-cases.
* Quando i test diventano ingovernabili a causa di dipendenze globali (allora isolo tramite dependency injection).

**11) Piano di rollout e rollback**

* Rollout:
  1. Merge su branch feature
  2. Deploy su staging con Firestore emulator / staging project
  3. Smoke tests (session create/join/order)
  4. Promozione su production
* Rollback:
  1. usare backup [backups](vscode-file://vscode-app/c:/Users/Anselmo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) per ripristino lib\_new se serve, ripristinare regole precedenti, revert commit.
* Nota: mantenere tag di rilascio e changelog.