

S M A R T



S H O P

B o s c a g l i a P i e r l u i g i

M i c h e l e C a s c i o n e

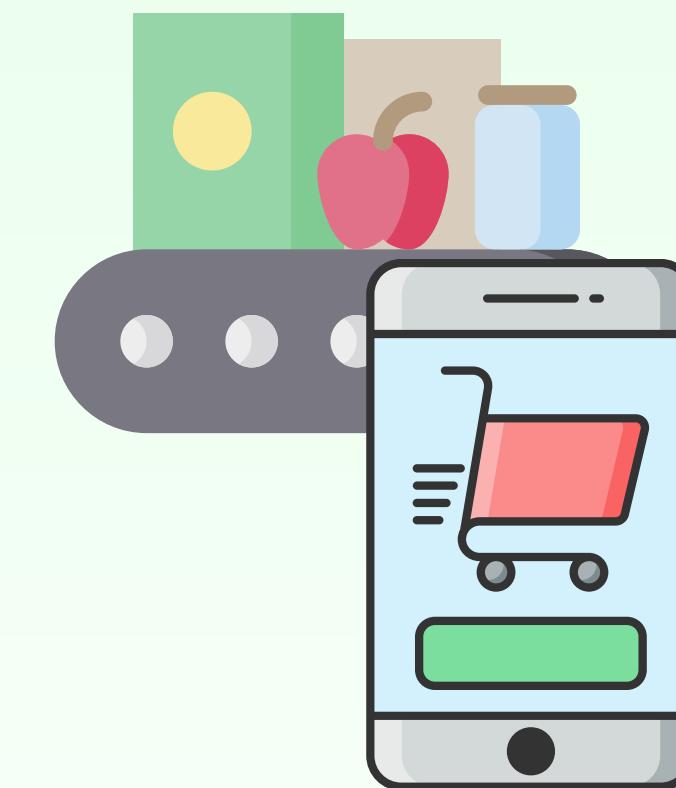
DIPENDENTE

Gestione
ordini



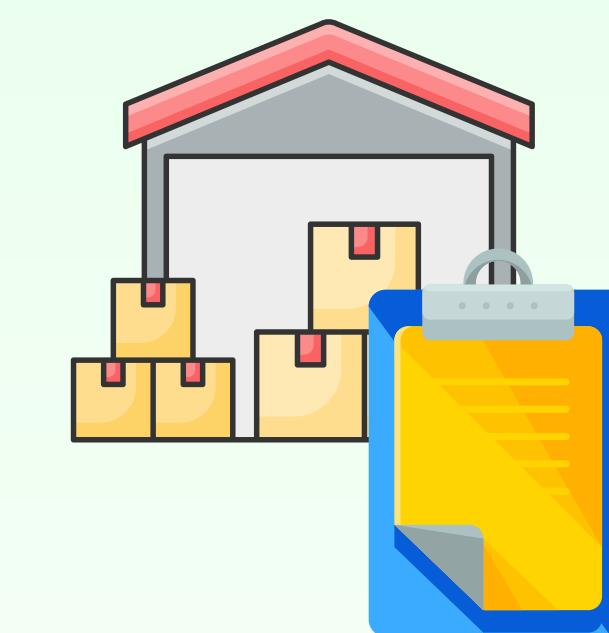
CLIENTE

Ritiro spesa: Locker
Consegna a domicilio
Spesa Online: Catalogo

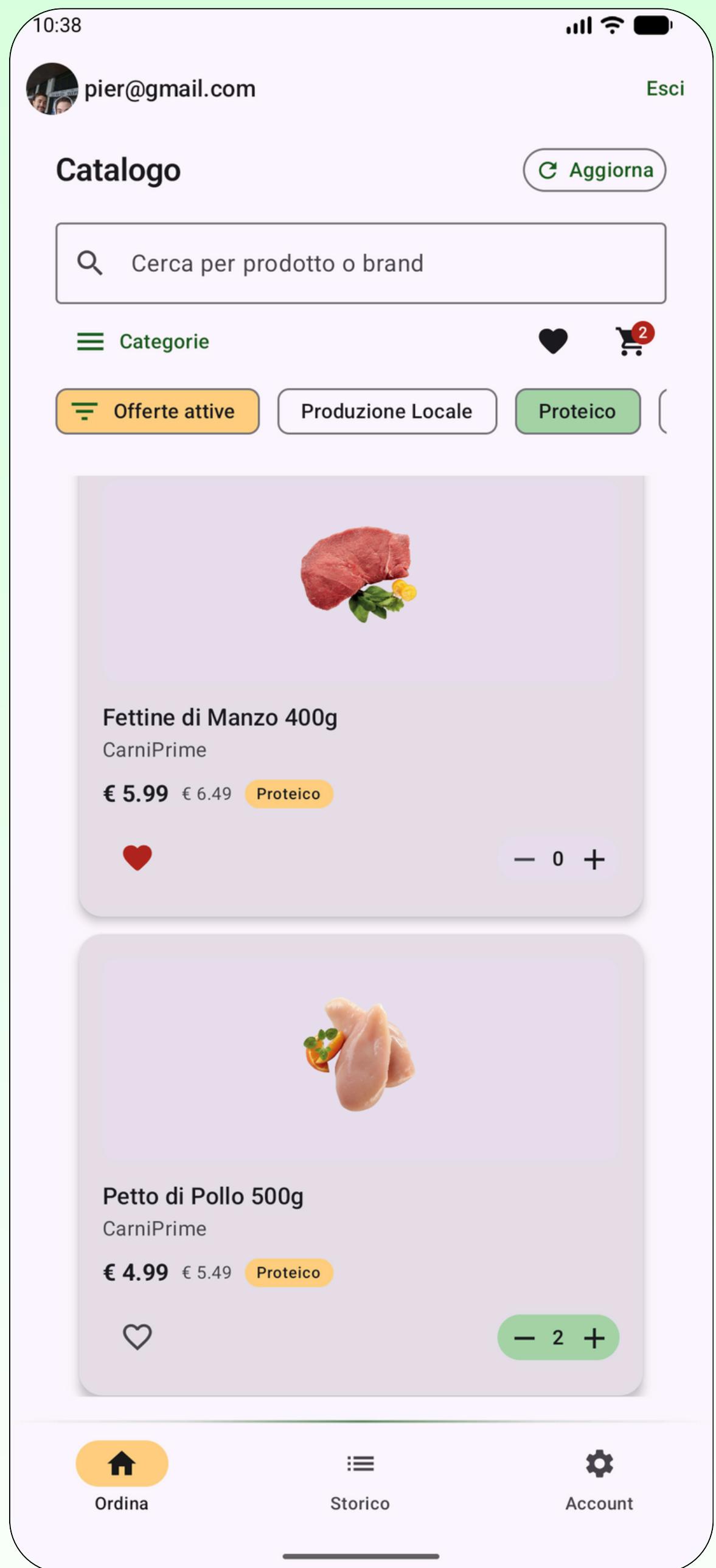


RESPONSABILE

Gestione
magazzino



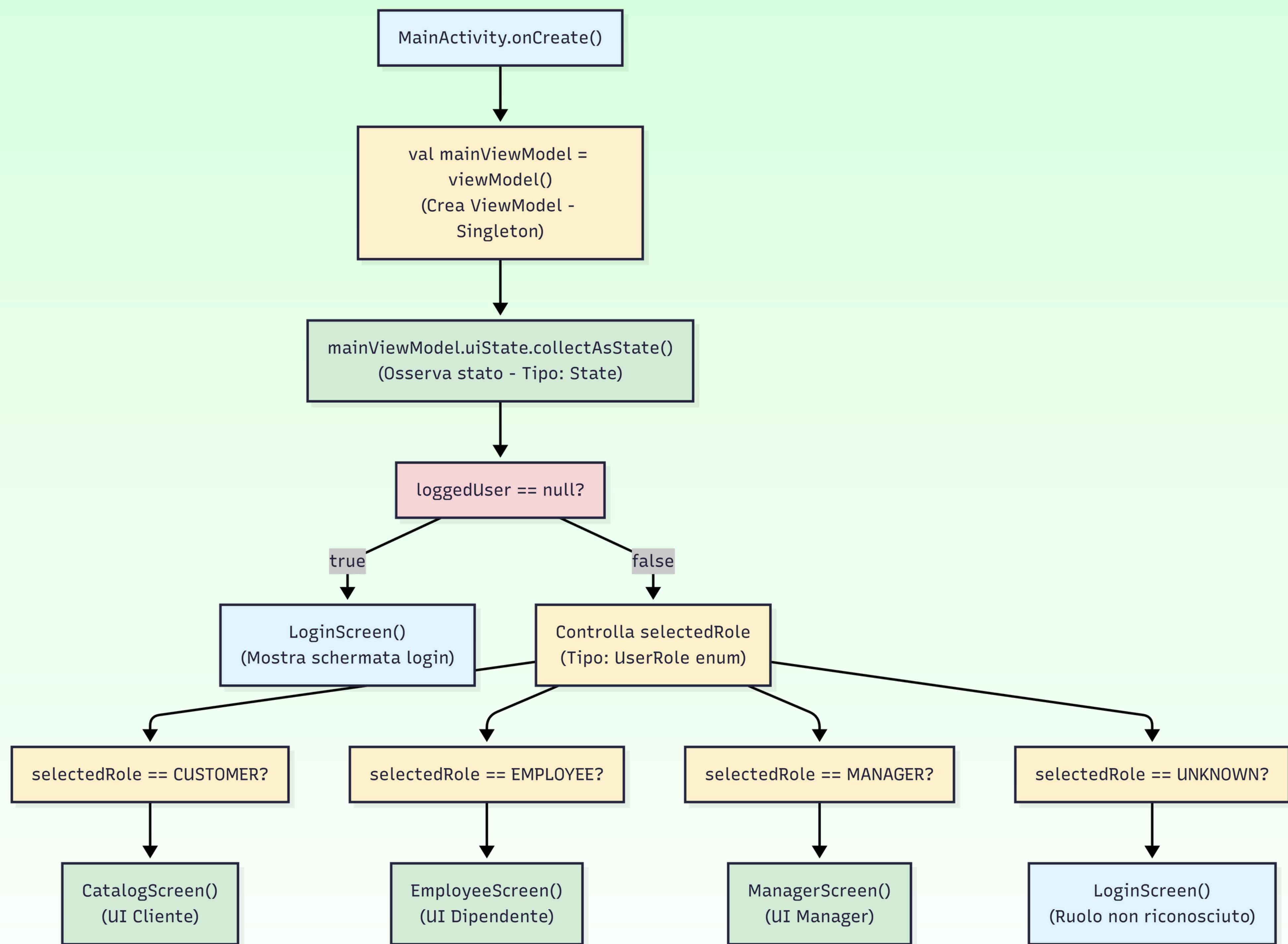
INTRODUZIONE: Obiettivi Iniziali



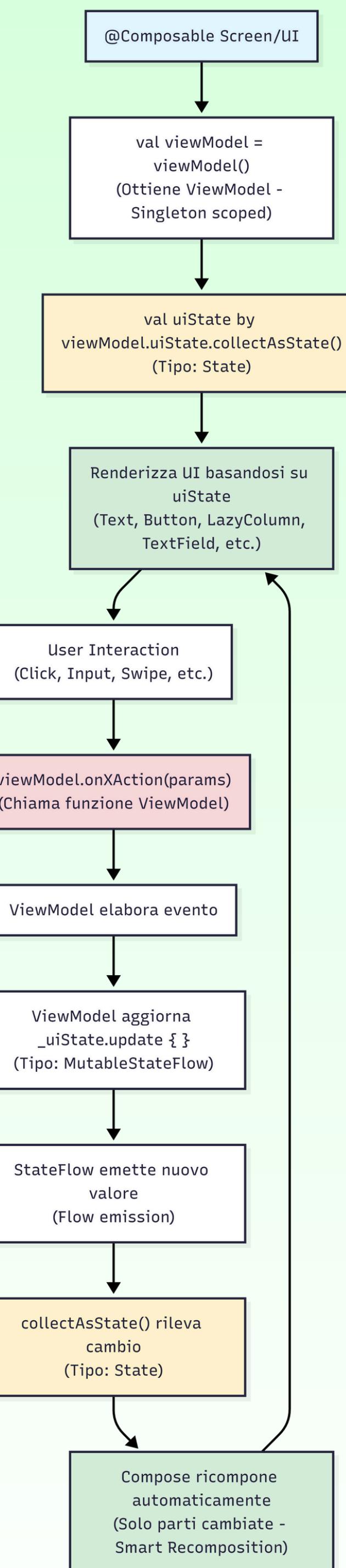
REQUISITI: Servizi Forniti



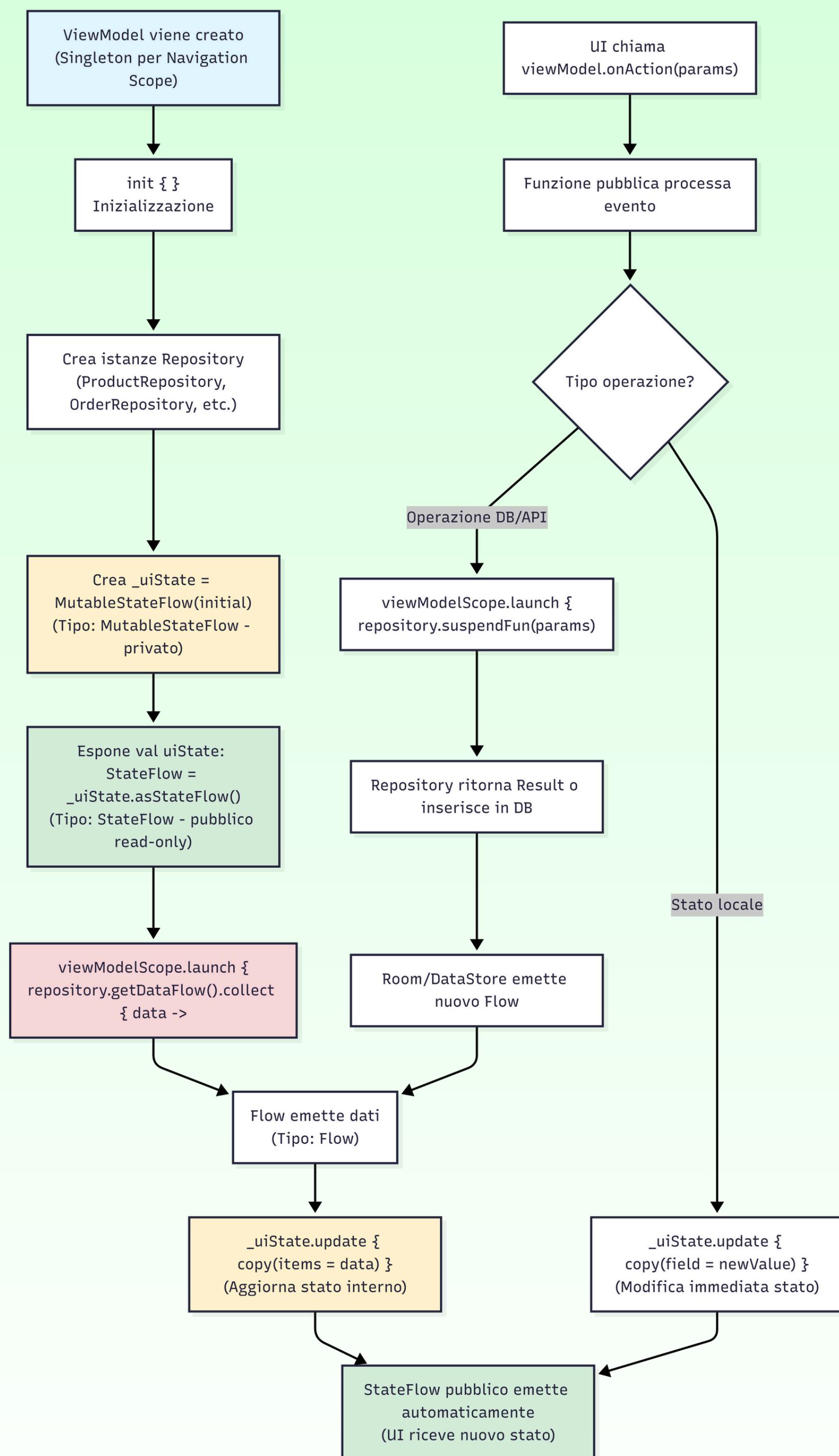
DESIGN



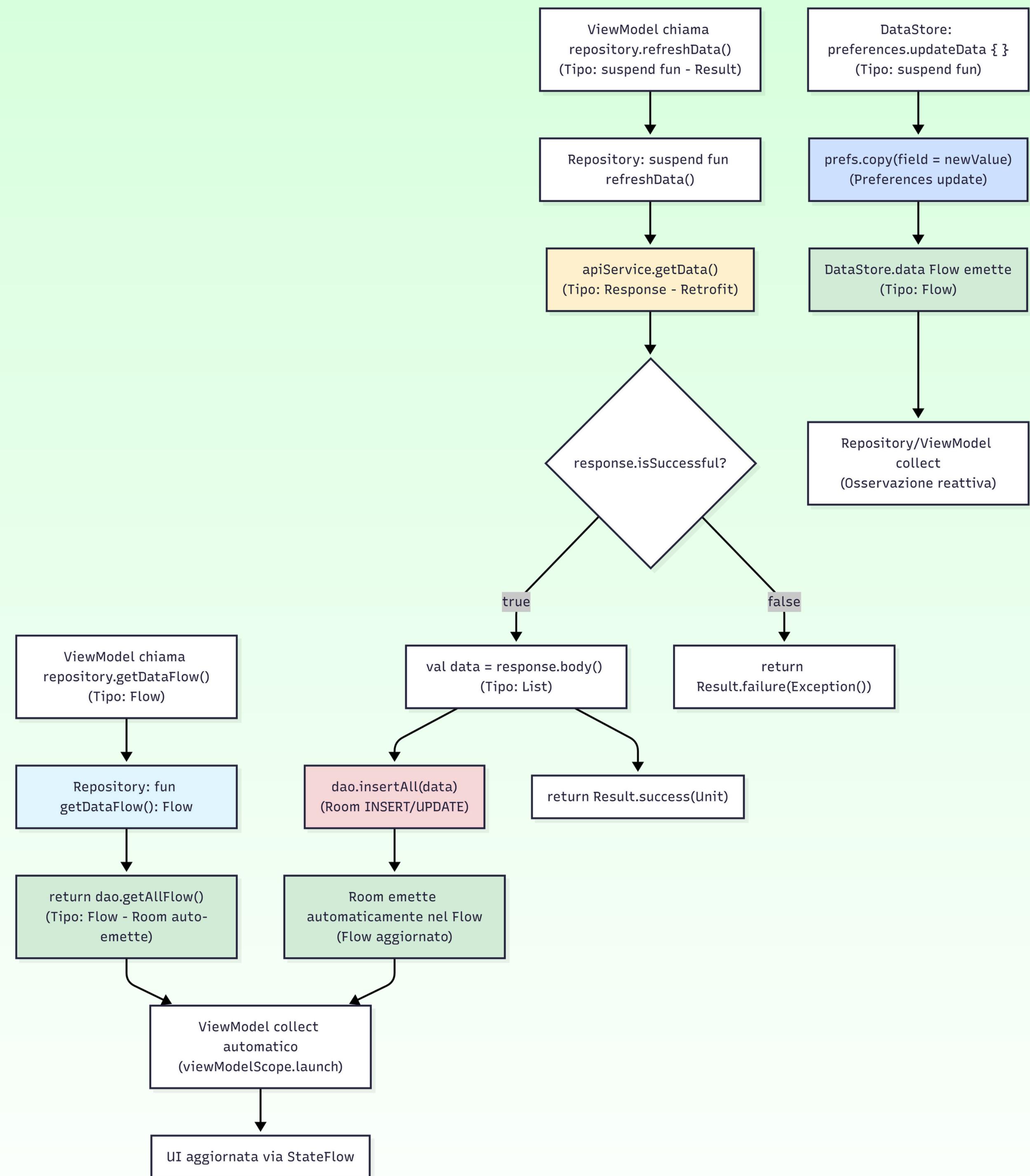
IMPLEMENTAZIONE : Entry Point



IMPLEMENTAZIONE: View Layer



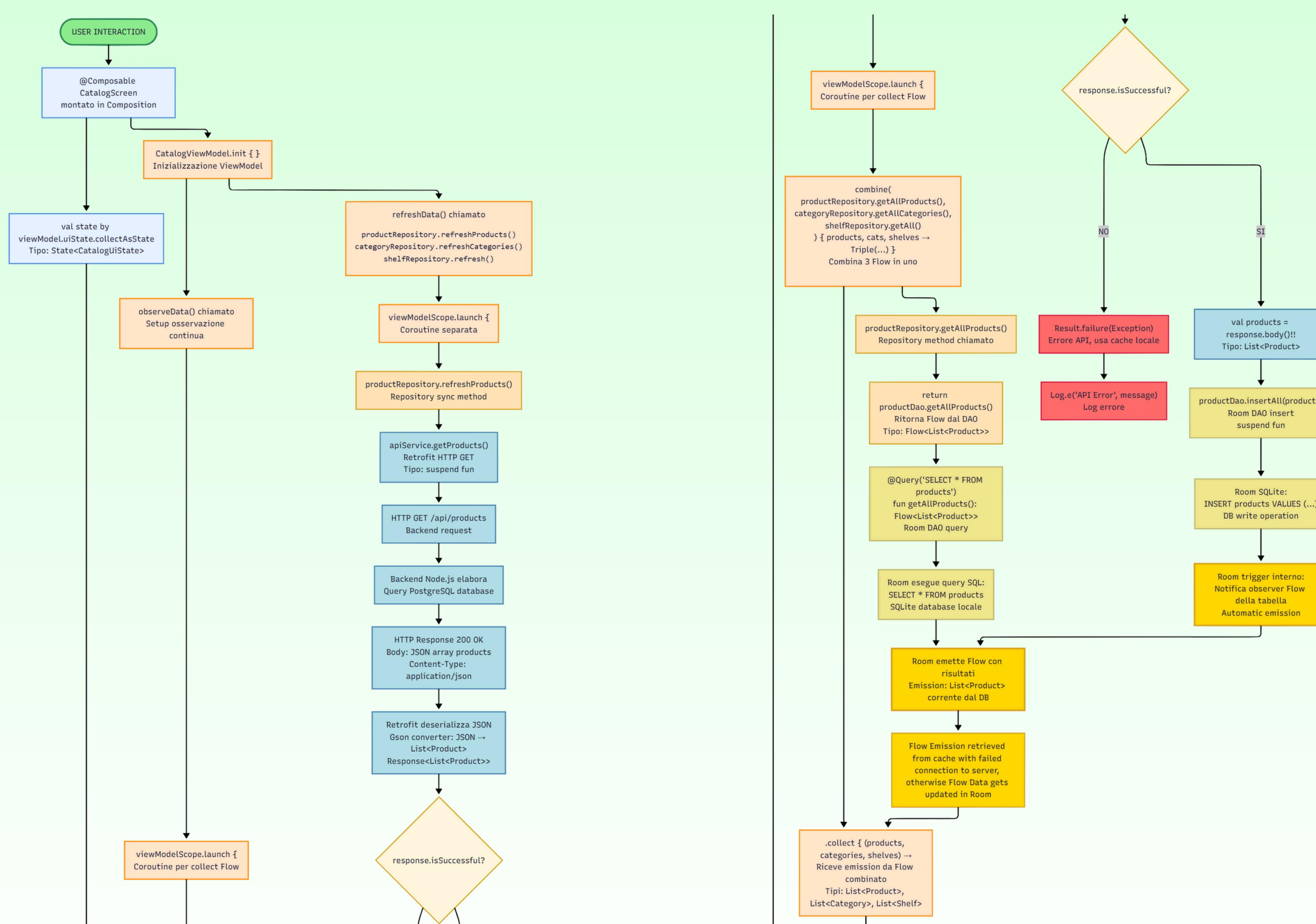
IMPLEMENTAZIONE: View Model Layer



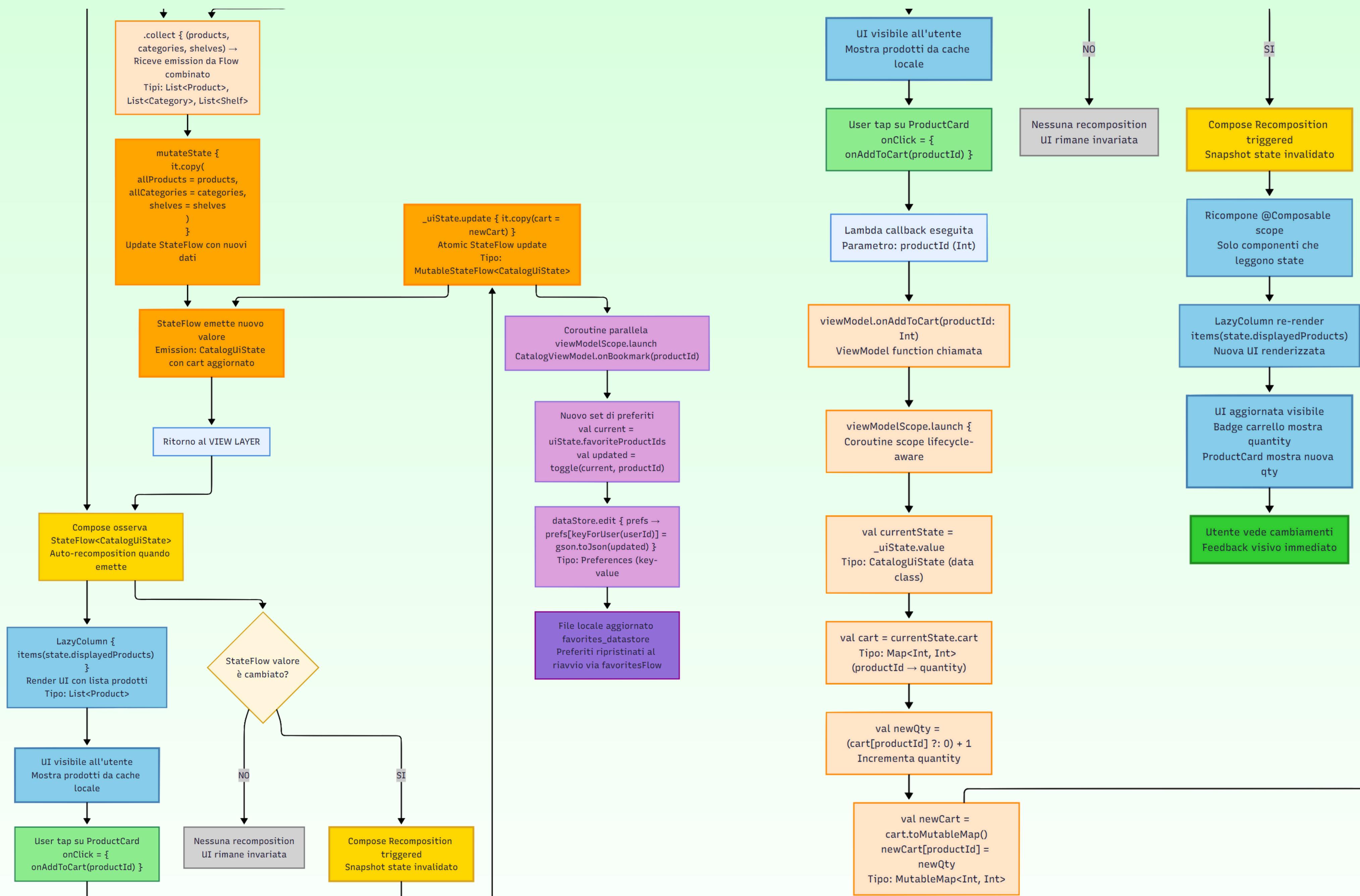
IMPLEMENTAZIONE : Model Layer



IMPLEMENTAZIONE: Model Layer Database



IMPLEMENTAZIONE: Flusso Ordine



IMPLEMENTAZIONE: Carrello e Preferiti

UnitViewModelTest

Verificano che la logica di business nei ViewModel funzioni correttamente in modo isolato, simulando tutte le dipendenze con mock. Controllano che gli stati dell'UI cambino correttamente, le chiamate ai Repository siano effettuate con i parametri corretti, gli errori vengano gestiti appropriatamente e i Flow emettano i valori attesi.

IntegrationTest

Verificano che più componenti dell'applicazione funzionino correttamente insieme, testando l'interazione completa tra API backend (simulato), Repository, Database locale (Room) e DataStore. Controllano che i dati viaggino correttamente dal backend fino al database locale e che i Flow emettano i valori corretti.

uiTest

Test strumentali che verificano l'intera applicazione dal punto di vista dell'utente, simulando interazioni reali (click, input, scroll) e testando flussi completi dall'inizio alla fine. Richiedono un dispositivo o emulatore Android per essere eseguiti.

JUnit: Framework base per scrivere ed eseguire test, fornisce annotazioni fondamentali (@Test, @Before, @After). [Unit & UI Test]

Kotlinx Coroutines Test: Strumenti per testare codice asincrono, permette di controllare il tempo nei test con runTest { } e advanceUntilIdle(). [UnitTest]

Robolectric: Permette di eseguire test Android sulla JVM senza emulatore, rendendo i test molto più veloci. [UnitTest]

MockWebServer (OkHttp): Crea un server HTTP finto che simula risposte del backend reale per testare chiamate API senza server vero. [Integration Test]

MockK: Libreria per creare oggetti mock che simulano il comportamento di dipendenze reali senza database o API. [UnitViewModel Test]

MainDispatcherRule (Custom): Regola personalizzata che rende le coroutine sincrone e gestite dal test per evitare errori di timing. [UnitViewModel Test]

Compose UI Test: Framework di test per applicazioni Jetpack Compose, permette di trovare elementi UI e simulare interazioni.

AndroidJUnit4: Runner di test che esegue i test su dispositivo/emulatore Android reale.

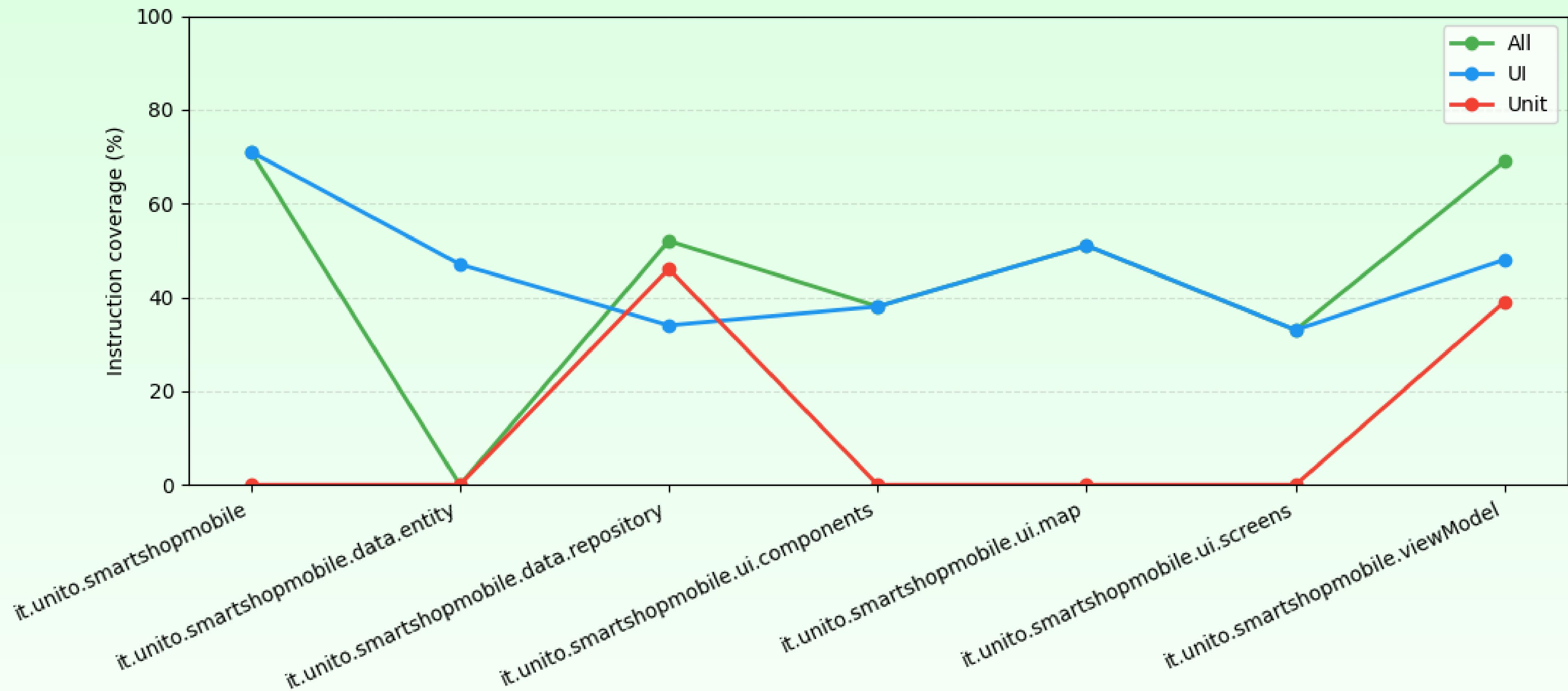
ComposeTestRule: Gestisce il ciclo di vita dell'Activity e fornisce accesso all'albero UI di Compose durante i test.

Semantic Matchers: Funzioni per trovare elementi UI in base a proprietà semantiche (testo, descrizione, azioni: `hasText()`, `hasContentDescription()`, `hasImeAction()`).

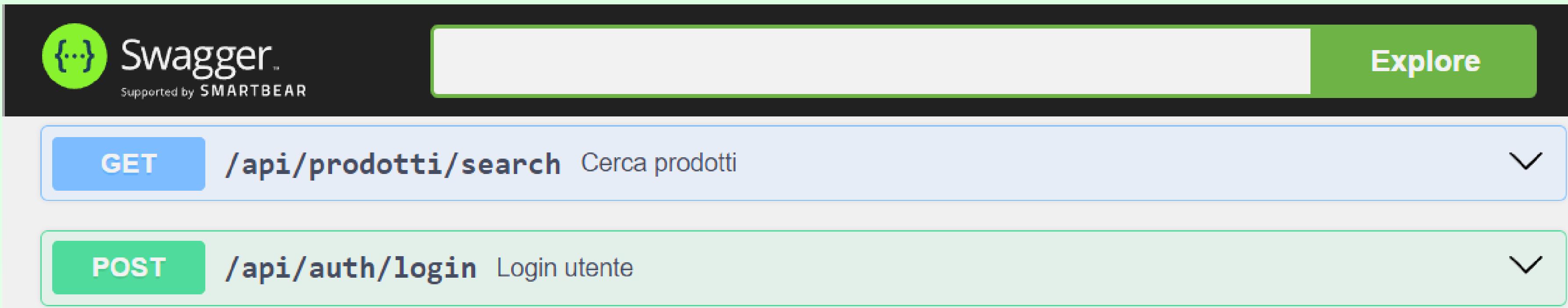
Test Actions: Azioni che simulano interazioni utente: `performClick()`, `performTextInput()`, `performScrollTo()`.

Test Assertions: Verifiche che controllano lo stato dell'UI: `assertExists()`, `assertIsDisplayed()`, `assertTextEquals()`.

waitForIdle/waitUntil: Funzioni per attendere che l'UI completi operazioni asincrone ed evitare errori di timing.



TEST : Jacoco Code Coverage



The image shows the Swagger UI interface for a RESTful API. At the top, there's a navigation bar with the 'Swagger' logo and the text 'Supported by SMARTBEAR'. On the right side of the bar is a green button labeled 'Explore'. Below the bar, there are two main sections. The first section, with a blue background, contains a 'GET' button followed by the endpoint '/api/prodotti/search' and the description 'Cerca prodotti'. The second section, with a green background, contains a 'POST' button followed by the endpoint '/api/auth/login' and the description 'Login utente'. Both sections have a small downward arrow icon on the right side.

```
@Composable
/**
 * Schermata di login MVVM che si appoggia a `LoginViewModel`.
 *
 * Visualizza il form di autenticazione, gestisce errori/toast tramite `SnackbarHost`
 * e propaga l'utente autenticato al chiamante via `onLoginSuccess`.
 *
 * @param viewModel ViewModel di login da cui leggere stato ed eseguire azioni
 * @param onLoginSuccess Callback invocato con utente e ruolo dopo login riuscito
 * @param modifier Modificatore Compose opzionale
 */
```