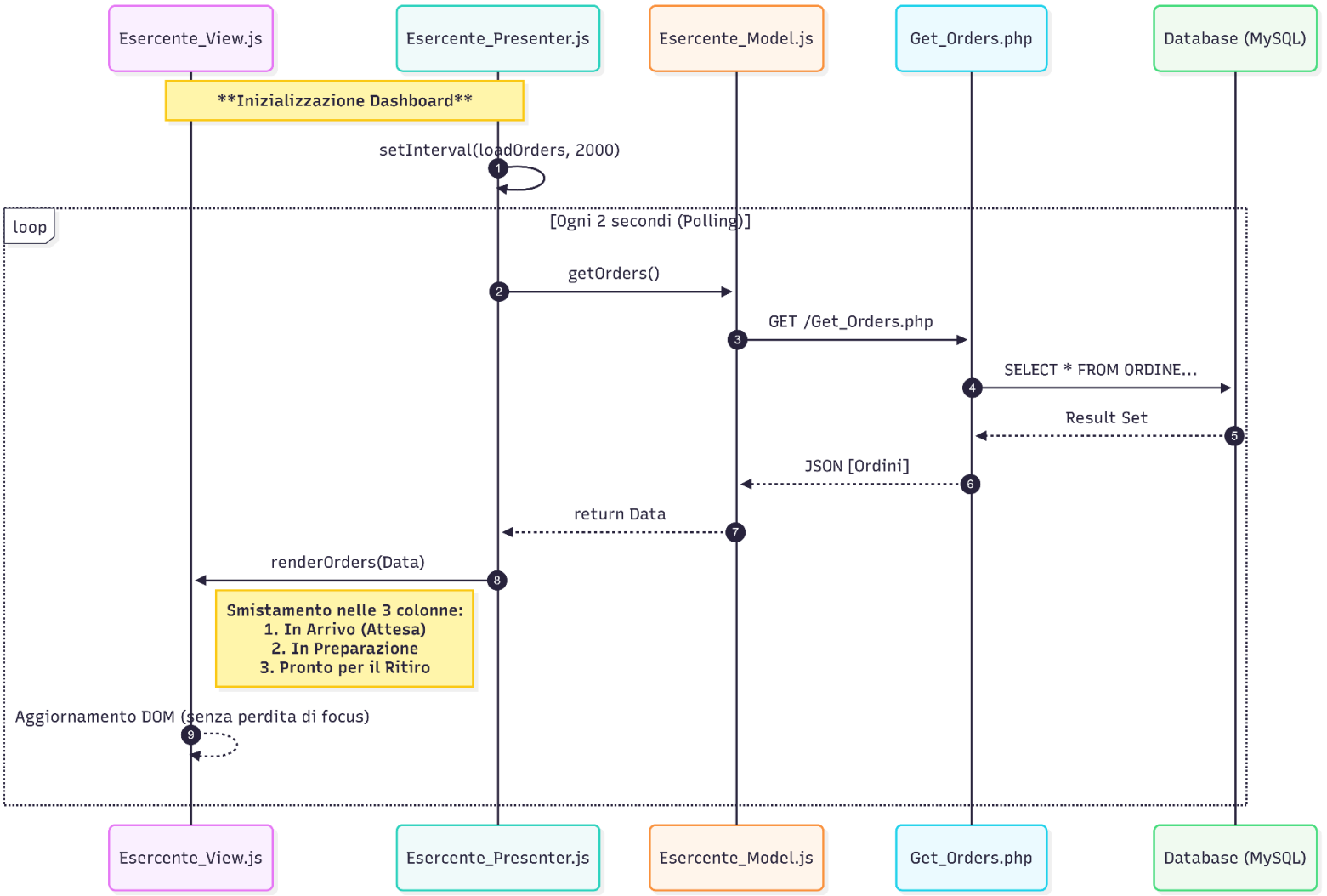
Diagramma di Sequenza - Ciclo di Vita Ordine e Polling (Task 3.4)

Il diagramma illustra il processo automatizzato di sincronizzazione tra il backend e l'interfaccia dell'esercente, garantendo che il flusso degli ordini sia sempre aggiornato senza intervento manuale.

1. **Inizializzazione Polling**: All'avvio del modulo, l'**Esercente_Presenter.js** imposta un timer ciclico (`setInterval`) con un intervallo di 2000ms.
2. **Richiesta Dati (Fetch)**: Ad ogni ciclo, il Presenter interroga l'**Esercente_Model.js** per ottenere la lista aggiornata degli ordini.
3. **Comunicazione API**: Il Model esegue una chiamata asincrona all'API **Get_Orders.php**, che interroga il Database per recuperare tutti gli ordini associati all'esercente loggato.
4. **Elaborazione Risposta**: I dati ricevuti in formato JSON vengono passati dal Model al Presenter.
5. **Rendering Dinamico (Kanban)**: Il Presenter invoca il metodo `renderOrders` della **Esercente_View.js**.
6. **Distribuzione nelle Colonne**: La View analizza lo stato di ogni ordine (`attesa`, `preparazione`, `pronto`) e inietta dinamicamente le card HTML nelle tre colonne corrispondenti della Dashboard: *In Arrivo*, *In Preparazione* e *Pronto per il Ritiro*.
7. **Mantenimento Focus**: La logica di rendering della View include un controllo sul focus degli input (come il campo OTP) per evitare che il refresh del polling interrompa l'inserimento di dati da parte dell'utente.


Esercente_View.js
Esercente_Presenter.js
Esercente_Model.js
Get_Orders.php
Database (MySQL)
Inizializzazione Dashboard
setInterval(loadOrders, 2000)
1
loop
[Ogni 2 secondi (Polling)]
getOrders()
2
GET /Get_Orders.php
3
SELECT * FROM ORDINE...
4
Result Set
5
JSON [Ordini]
6
return Data
7
renderOrders(Data)
8
Smistamento nelle 3 colonne:
1. In Arrivo (Attesa)
2. In Preparazione
3. Pronto per il Ritiro
Aggiornamento DOM (senza perdita di focus)
9
Esercente_View.js
Esercente_Presenter.js
Esercente_Model.js
Get_Orders.php
Database (MySQL)