**Design dell'Architettura: Progetto "Fantavacanze"**

* **Architettura Utilizzata: CLEAN Architecture**

Per lo sviluppo di "Fantavacanze" verrà utilizzata la **CLEAN Architecture**, che si basa sulla separazione dei diversi livelli di logica e responsabilità all'interno dell'applicazione. Questa scelta permette di ottenere un codice più manutenibile, modulare e testabile, facilitando gli aggiornamenti futuri e la collaborazione tra più sviluppatori.

La **CLEAN Architecture** prevede la suddivisione dell'app in quattro livelli principali:

* **Presentation Layer**: Questa parte gestisce l'interfaccia utente e la logica di presentazione. Per la gestione della logica di presentazione verrà utilizzato **flutter\_bloc**, che consente di mantenere separata la logica di business dallo stato dell'interfaccia utente. La logica di presentazione sarà gestita tramite i vari Cubit/Bloc che si occuperanno di inviare e ricevere stati e eventi, mantenendo l'interfaccia utente reattiva e coerente.
* **Domain Layer**: Include le entità, gli use case e le interfacce necessarie per definire le regole di business dell'applicazione. Gli **use case** rappresentano le diverse funzionalità dell'app (come la creazione di una lega o l'invio di una sfida). Le **entità** rappresentano gli oggetti principali dell'app, come "User", "League", "Challenge".
* **Data Layer**: Si occupa della gestione dei dati e delle interazioni con il database o altre fonti di dati (es. API esterne). La connessione al database sarà gestita tramite Supabase, e verrà implementato un repository pattern che si occuperà di fornire accesso ai dati necessari al Domain Layer. Inoltre, questo livello gestirà il local data source per i dati salvati e da salvare utilizzando i box di Hive. Questo livello è responsabile della serializzazione e deserializzazione dei dati per renderli utilizzabili dal Domain Layer.
* **Infrastructure Layer**: Include servizi esterni, package e implementazioni che sono indipendenti dalle regole di business. Ad esempio, il servizio di autenticazione tramite Supabase, l'integrazione con RevenueCat per i pagamenti in-app, e la gestione del caching offline tramite Hive.
* **Linguaggi di Programmazione Utilizzati**
* **Dart**: Verrà utilizzato Dart come linguaggio principale, dato che Flutter è basato su Dart. Dart offre ottime prestazioni e consente di creare un'unica base di codice per entrambe le piattaforme, Android e iOS.
* **Package Esterni Utilizzati**
* **flutter\_bloc**: Utilizzato per gestire lo stato dell'applicazione, seguendo il pattern Bloc/Cubit che permette di separare la logica di presentazione dalla UI.
* **supabase\_flutter**: Utilizzato per l'interazione con Supabase, che sarà la base per il backend, il database e l'autenticazione degli utenti.
* **uuid**: Utilizzato per generare identificatori unici per le varie entità, come ad esempio le leghe o le sfide.
* **internet\_connection\_checker\_plus**: Utilizzato per verificare la connessione internet, garantendo una migliore esperienza utente in caso di problemi di connettività e per mostrare i dati salvati con Hive anziché effettuare chiamate al database quando possibile.
* **image\_picker**: Utilizzato per consentire agli utenti di scattare foto e caricarle all'interno dell'applicazione, ad esempio nella sezione dei ricordi.
* **fpdart**: Utilizzato per la gestione di operazioni asincrone e la manipolazione funzionale dei dati, aumentando la robustezza e la leggibilità del codice.
* **get\_it**: Utilizzato per la gestione delle dipendenze, permettendo di separare le dipendenze tra i vari layer dell'app in modo pulito e modulare.
* **hive**: Utilizzato per il caching offline, consentendo agli utenti di continuare a utilizzare l'app anche in assenza di connessione.
* **intl**: Utilizzato per la formattazione delle date e la localizzazione, permettendo all'app di essere facilmente adattata a diverse lingue e formati regionali.
* **RevenueCat**: Utilizzato per la gestione dei pagamenti in-app, inclusi abbonamenti e acquisti una tantum, garantendo la conformità agli standard PCI-DSS e semplificando l'implementazione della monetizzazione.
* **Visione d'Insieme dei Moduli e delle Entità**

L'app sarà suddivisa in diversi moduli per garantire una buona manutenibilità e un'organizzazione efficiente del codice. Ogni modulo rappresenta una parte distinta delle funzionalità dell'app.

* **Auth Module**:
* **Entità**: User, Session
* **Descrizione**: Questo modulo si occupa della registrazione, autenticazione e gestione delle sessioni degli utenti. Interagisce direttamente con Supabase per la creazione di nuovi utenti e la gestione del login/logout.
* **League Module**:
* **Entità**: League, Participant, TeamParticipant, IndividualParticipant
* **Descrizione**: Gestisce la creazione e la gestione delle leghe. Gli utenti possono creare nuove leghe, unirsi a leghe esistenti tramite codice, e gestire le regole di gioco. Le entità Participant e le sue sottoclassi rappresentano i singoli partecipanti (individuali o a squadre).
* **Challenge Module**:
* **Entità**: QuickChallenge, ChallengeEvent
* **Descrizione**: Gestisce la creazione e la partecipazione alle sfide veloci. L'entità QuickChallenge rappresenta una sfida temporanea e gli utenti possono partecipare tramite invito. ChallengeEvent rappresenta ogni evento legato a una sfida.
* **Games Module**:
* **Entità**: Game
* **Descrizione**: Questo modulo include i giochi alcolici come "Dare to Share", "5 Seconds Rule" e "Speedy Brains". I giochi sono configurati con un set di regole predefinito che può essere estratto da Supabase o generato tramite ChatGPT API.
* **Memory Module**:
* **Entità**: Memory, Album
* **Descrizione**: Gestisce le fotografie scattate dagli utenti durante le varie sfide e leghe. Gli utenti possono salvare i loro ricordi e creare album che possono essere successivamente stampati.
* **Store Module**:
* **Entità**: Merchandise, Order
* **Descrizione**: Gestisce l'acquisto di prodotti fisici, come il merchandise di Fantavacanze. Interagisce con RevenueCat per la gestione dei pagamenti in-app.
* **Interazioni col Database**
* **Supabase** sarà utilizzato come backend e database per conservare i dati relativi agli utenti, alle leghe, alle sfide e agli eventi di gioco. Tutte le entità avranno una rappresentazione nel database, e le interazioni saranno mediate tramite repository nel Data Layer per garantire una separazione tra la logica di business e la persistenza dei dati.
* Ogni modulo interagirà con Supabase tramite repository dedicati che forniranno metodi per eseguire operazioni CRUD (Create, Read, Update, Delete). Ad esempio, il LeagueRepository gestirà la creazione, aggiornamento e recupero delle leghe, mentre il UserRepository si occuperà della gestione dei dati degli utenti.
* **Conclusione**

L'utilizzo della CLEAN Architecture, insieme a flutter\_bloc e a una serie di package esterni ben selezionati, permette di ottenere un'app strutturata, facilmente manutenibile e scalabile. I singoli moduli definiscono responsabilità chiare, e la separazione tra logica di presentazione, dominio e dati assicura che il sistema possa evolversi senza creare dipendenze indesiderate. Le interazioni con Supabase tramite repository e l'integrazione con RevenueCat per la monetizzazione rendono il sistema efficiente e pronto a scalare.