**ENVISIONING GENERALE PROGETTO P.A.C.**

**Architettura**

1. **Logica in Java**:
   * Usa un server Java (ad esempio Spring Boot) per gestire la logica applicativa e l'interazione con il database.
   * Il modello MVC sarà implementato nel backend con Spring Boot:
     + **Model**: Rappresenta i dati e la logica aziendale.
     + **View**: Fornisce dati JSON/REST per le interfacce Flutter.
     + **Controller**: Gestisce le richieste HTTP e invia i dati alla View.
2. **Flutter per la grafica**:
   * Sviluppa interfacce responsive utilizzando Flutter. Questo ti permette di creare un'app unica per mobile (Android e iOS) e desktop (Windows, macOS e Linux).
3. **Comunicazione client-server**:
   * Utilizza API REST o WebSocket per la comunicazione tra il server Java e il client Flutter.

**Installazione senza dipendenze aggiuntive**

1. **Desktop**:
   * **Windows**:
     + Usa **Flutter Desktop** per creare l'applicazione nativa.
     + Per distribuire senza dipendenze aggiuntive, puoi impacchettare l'app con uno strumento come **MSIX Packaging Tool** o **Inno Setup** per Windows.
   * **Server Backend**:
     + Puoi distribuire il server backend come un file JAR o WAR eseguibile (Spring Boot offre la possibilità di creare JAR autonomi).
2. **Mobile**:
   * Crea pacchetti APK per Android e IPA per iOS utilizzando Flutter.
   * Gli utenti non avranno bisogno di installare dipendenze aggiuntive se impacchetti l'app correttamente.

**Docker**

Docker è utile per:

* **Server Backend**:
  + Contenere il server Java in un'immagine Docker. Questo assicura che funzioni su qualsiasi macchina che supporti Docker, indipendentemente dalle dipendenze locali.
* **Distribuzione facilitata**:
  + Puoi impacchettare l'intero backend con Docker e distribuire un'immagine pronta per l'esecuzione.

Non è necessario Docker per il client (Flutter), ma è consigliato per la parte server, in particolare per semplificare la distribuzione su diversi sistemi operativi o ambienti.

**Workflow di sviluppo**

1. **Backend**:
   * Sviluppa il server Java con Spring Boot.
   * Implementa API REST o WebSocket.
   * Contenitore Docker per il server.
2. **Frontend**:
   * Sviluppa l'interfaccia grafica con Flutter.
   * Integra le chiamate API per la comunicazione con il server.
3. **Distribuzione**:
   * Usa Docker per il backend.
   * Impacchetta il client con strumenti nativi di Flutter per Android, iOS, e desktop.

Creare app desktop/android client server e bisogna avere 3 eseguibili: programma server, app utente e app gestore (che inserisce i dati).

. SERVER: programma da accendere che contiene la logica java e fa comunicare le 2 app.

. APP GESTORE: flutter che permette di inserire nome citta e liste da riempire con luoghi da visitare, luoghi di riposo e luoghi di ristoro, con orari apertura chiusura e prezzi.

. APP UTENTE: flutter che permette all’utente di inserire città, luoghi da visitare, budget disponibile e tempo previsto.

Gestione del percorso migliore con grafi.

GRAFICA:

design semplice e bianco (alla pokemon) con linee del colore del logo.

Proposte nome:

. OpTripmize, OpTrip (optimize + trip)

. altro? Con boost, improve, …

Proposte logo:

. icona posizione, valigia, omino turista, viaggio

Toolchain

GitHub, Java, Flutter, Spring Boot, Docker, SQLite

Sistema per la gestione di visite turistiche a luoghi di interesse e centri storici.