Flight Booking Chatbot Implementation Guide - SerpApi Integration

Overview

Convert the existing "Chat SDK" AI chatbot into a comprehensive flight booking agent using SerpApi Google Flights API. This guide provides detailed technical instructions for implementation.

Prerequisites

- Existing Next.js 15 + Vercel AI SDK chatbot codebase
- SerpApi account and API key
- Access to existing Supabase database
- Understanding of the current function calling architecture

1. Environment Setup

1.1 Install Required Dependencies

```
npm install @types/node
# No additional packages needed - SerpApi uses standard fetch
```

1.2 Environment Variables

Add to `.env.local`:

```
SERPAPI_KEY=your_serpapi_key_here
SERPAPI_BASE_URL=https://serpapi.com/search.json
```

1.3 SerpApi Account Setup

1. Sign up at https://serpapi.com

2. Get API key from dashboard
3. Start with free tier (100 searches/month)
4. Upgrade to paid plan ($50/month for 5000 searches) when needed

2. Database Schema Updates

2.1 New Tables for Flight Data

```
-- Add to existing Supabase database

-- Flight searches table to store user search history
CREATE TABLE flight_searches (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  user_id UUID REFERENCES "User"(id) ON DELETE CASCADE,
  chat_id UUID REFERENCES "Chat"(id) ON DELETE CASCADE,
  origin_code VARCHAR(3) NOT NULL,
  destination_code VARCHAR(3) NOT NULL,
  departure_date DATE NOT NULL,
  return_date DATE,
  passengers INTEGER DEFAULT 1,
  travel_class VARCHAR(20) DEFAULT 'economy',
  trip_type VARCHAR(20) DEFAULT 'one-way', -- one-way, round-trip
  search_results JSONB, -- Store SerpApi response
  created_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, N
OW()) NOT NULL,
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, N
OW()) NOT NULL
);

-- Price alerts table for price monitoring
CREATE TABLE price_alerts (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  user_id UUID REFERENCES "User"(id) ON DELETE CASCADE,
  flight_search_id UUID REFERENCES flight_searches(id) ON DELETE CASCAD
E,
```

```
  target_price DECIMAL(10,2),
  current_price DECIMAL(10,2),
  is_active BOOLEAN DEFAULT true,
  alert_type VARCHAR(20) DEFAULT 'price_drop', -- price_drop, price_increase
  created_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, N
OW()) NOT NULL,
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, N
OW()) NOT NULL
);

-- Airport codes reference table
CREATE TABLE airports (
  code VARCHAR(3) PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  city VARCHAR(100) NOT NULL,
  country VARCHAR(100) NOT NULL,
  timezone VARCHAR(50),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, N
OW()) NOT NULL
);

-- Add indexes
CREATE INDEX idx_flight_searches_user_id ON flight_searches(user_id);
CREATE INDEX idx_flight_searches_chat_id ON flight_searches(chat_id);
CREATE INDEX idx_price_alerts_user_id ON price_alerts(user_id);
CREATE INDEX idx_airports_city ON airports(city);
CREATE INDEX idx_airports_name ON airports(name);
```

2.2 Drizzle Schema Updates

Add to `lib/db/schema.ts`:

```
import { pgTable, uuid, varchar, date, integer, jsonb, decimal, boolean, timesta
mp, index } from 'drizzle-orm/pg-core';

export const flightSearches = pgTable('flight_searches', {
```

```
  id: uuid('id').defaultRandom().primaryKey(),
  userId: uuid('user_id').references(() => user.id, { onDelete: 'cascade' }),
  chatId: uuid('chat_id').references(() => chat.id, { onDelete: 'cascade' }),
  originCode: varchar('origin_code', { length: 3 }).notNull(),
  destinationCode: varchar('destination_code', { length: 3 }).notNull(),
  departureDate: date('departure_date').notNull(),
  returnDate: date('return_date'),
  passengers: integer('passengers').default(1),
  travelClass: varchar('travel_class', { length: 20 }).default('economy'),
  tripType: varchar('trip_type', { length: 20 }).default('one-way'),
  searchResults: jsonb('search_results'),
  createdAt: timestamp('created_at').defaultNow().notNull(),
  updatedAt: timestamp('updated_at').defaultNow().notNull(),
}, (table) => ({
  userIdIdx: index('idx_flight_searches_user_id').on(table.userId),
  chatIdIdx: index('idx_flight_searches_chat_id').on(table.chatId),
}));

export const priceAlerts = pgTable('price_alerts', {
  id: uuid('id').defaultRandom().primaryKey(),
  userId: uuid('user_id').references(() => user.id, { onDelete: 'cascade' }),
  flightSearchId: uuid('flight_search_id').references(() => flightSearches.id, { on
Delete: 'cascade' }),
  targetPrice: decimal('target_price', { precision: 10, scale: 2 }),
  currentPrice: decimal('current_price', { precision: 10, scale: 2 }),
  isActive: boolean('is_active').default(true),
  alertType: varchar('alert_type', { length: 20 }).default('price_drop'),
  createdAt: timestamp('created_at').defaultNow().notNull(),
  updatedAt: timestamp('updated_at').defaultNow().notNull(),
});

export const airports = pgTable('airports', {
  code: varchar('code', { length: 3 }).primaryKey(),
  name: varchar('name', { length: 255 }).notNull(),
  city: varchar('city', { length: 100 }).notNull(),
  country: varchar('country', { length: 100 }).notNull(),
```

```
  timezone: varchar('timezone', { length: 50 }),
  createdAt: timestamp('created_at').defaultNow().notNull(),
});
```

3. SerpApi Integration Service

3.1 Create SerpApi Service

Create `lib/services/serpapi.ts`:

```
import { z } from 'zod';

// Type definitions for SerpApi Google Flights responses
export const FlightSchema = z.object({
  departure_airport: z.object({
    name: z.string(),
    id: z.string(),
    time: z.string(),
  }),
  arrival_airport: z.object({
    name: z.string(),
    id: z.string(),
    time: z.string(),
  }),
  duration: z.number(),
  airplane: z.string().optional(),
  airline: z.string(),
  airline_logo: z.string().optional(),
  travel_class: z.string(),
  flight_number: z.string(),
  legroom: z.string().optional(),
  extensions: z.array(z.string()).optional(),
  carbon_emissions: z.object({
    this_flight: z.number().optional(),
    typical_for_this_route: z.number().optional(),
```

```
    difference_percent: z.number().optional(),
  }).optional(),
  price: z.number().optional(),
});

export const FlightSearchResultSchema = z.object({
  best_flights: z.array(FlightSchema).optional(),
  other_flights: z.array(FlightSchema).optional(),
  price_insights: z.object({
    lowest_price: z.number().optional(),
    price_level: z.string().optional(), // "low", "typical", "high"
    typical_price_range: z.array(z.number()).optional(),
    price_history: z.array(z.object({
      date: z.string(),
      price: z.number(),
    })).optional(),
  }).optional(),
  search_metadata: z.object({
    id: z.string(),
    status: z.string(),
    json_endpoint: z.string(),
    created_at: z.string(),
    processed_at: z.string(),
    google_flights_url: z.string(),
    total_time_taken: z.number(),
  }),
  search_parameters: z.object({
    engine: z.string(),
    departure_id: z.string(),
    arrival_id: z.string(),
    outbound_date: z.string(),
    return_date: z.string().optional(),
    travel_class: z.string().optional(),
    adults: z.number().optional(),
  }),
});
```

```
export type FlightSearchResult = z.infer<typeof FlightSearchResultSchema>;
export type Flight = z.infer<typeof FlightSchema>;

export interface FlightSearchParams {
  departure_id: string;
  arrival_id: string;
  outbound_date: string;
  return_date?: string;
  travel_class?: 'economy' | 'premium_economy' | 'business' | 'first';
  adults?: number;
  children?: number;
  infants_in_seat?: number;
  infants_on_lap?: number;
  type?: 1 | 2 | 3; // 1: round-trip, 2: one-way, 3: multi-city
}

class SerpApiService {
  private apiKey: string;
  private baseUrl: string;

  constructor() {
    this.apiKey = process.env.SERPAPI_KEY!;
    this.baseUrl = process.env.SERPAPI_BASE_URL || 'https://serpapi.com/sear
ch.json';

    if (!this.apiKey) {
      throw new Error('SERPAPI_KEY environment variable is required');
    }
  }

  async searchFlights(params: FlightSearchParams): Promise<FlightSearchRes
ult> {
    const searchParams = new URLSearchParams({
      engine: 'google_flights',
      api_key: this.apiKey,
```

```
      departure_id: params.departure_id,
      arrival_id: params.arrival_id,
      outbound_date: params.outbound_date,
      ...(params.return_date && { return_date: params.return_date }),
      ...(params.travel_class && { travel_class: params.travel_class }),
      ...(params.adults && { adults: params.adults.toString() }),
      ...(params.children && { children: params.children.toString() }),
      ...(params.infants_in_seat && { infants_in_seat: params.infants_in_seat.toSt
ring() }),
      ...(params.infants_on_lap && { infants_on_lap: params.infants_on_lap.toStri
ng() }),
      ...(params.type && { type: params.type.toString() }),
    });

    try {
      const response = await fetch(`${this.baseUrl}?${searchParams.toString()}
`, {
        method: 'GET',
        headers: {
          'User-Agent': 'FlightBookingChatbot/1.0',
        },
      });

      if (!response.ok) {
        throw new Error(`SerpApi request failed: ${response.status} ${response.
statusText}`);
      }

      const data = await response.json();

      // Validate response with Zod
      const validatedData = FlightSearchResultSchema.parse(data);
      return validatedData;
    } catch (error) {
      console.error('SerpApi flight search error:', error);
      throw new Error(`Flight search failed: ${error instanceof Error ? error.mess
```

```
age : 'Unknown error'}`);
    }
  }

  async getFlightPrice(
    departure_id: string,
    arrival_id: string,
    outbound_date: string,
    return_date?: string
  ): Promise<{ lowest_price?: number; price_level?: string }> {
    const result = await this.searchFlights({
      departure_id,
      arrival_id,
      outbound_date,
      return_date,
      type: return_date ? 1 : 2, // round-trip or one-way
    });

    return {
      lowest_price: result.price_insights?.lowest_price,
      price_level: result.price_insights?.price_level,
    };
  }
}

export const serpApiService = new SerpApiService();
```

3.2 Airport Code Service

Create `lib/services/airports.ts`:

```
import { db } from '@/lib/db';
import { airports } from '@/lib/db/schema';
import { eq, ilike, or } from 'drizzle-orm';

// Common airport codes for quick reference
```

```
export const COMMON_AIRPORTS = {
  // North America
  'new york': ['JFK', 'LGA', 'EWR'],
  'nyc': ['JFK', 'LGA', 'EWR'],
  'los angeles': ['LAX'],
  'la': ['LAX'],
  'chicago': ['ORD', 'MDW'],
  'san francisco': ['SFO'],
  'sf': ['SFO'],
  'miami': ['MIA'],
  'boston': ['BOS'],
  'seattle': ['SEA'],
  'denver': ['DEN'],
  'atlanta': ['ATL'],
  'dallas': ['DFW', 'DAL'],
  'washington': ['DCA', 'IAD', 'BWI'],
  'dc': ['DCA', 'IAD', 'BWI'],

  // Europe
  'london': ['LHR', 'LGW', 'STN', 'LTN'],
  'paris': ['CDG', 'ORY'],
  'amsterdam': ['AMS'],
  'frankfurt': ['FRA'],
  'madrid': ['MAD'],
  'rome': ['FCO', 'CIA'],
  'barcelona': ['BCN'],
  'berlin': ['BER'],
  'zurich': ['ZUR'],
  'vienna': ['VIE'],

  // Asia
  'tokyo': ['NRT', 'HND'],
  'beijing': ['PEK', 'PKX'],
  'shanghai': ['PVG', 'SHA'],
  'hong kong': ['HKG'],
  'singapore': ['SIN'],
```

```
  'dubai': ['DXB'],
  'mumbai': ['BOM'],
  'delhi': ['DEL'],
  'bangkok': ['BKK'],
  'seoul': ['ICN', 'GMP'],
};

export class AirportService {
  /**
   * Find airport code from natural language input
   */
  static findAirportCode(input: string): string | null {
    const normalized = input.toLowerCase().trim();

    // Check if it's already a valid airport code
    if (/^[A-Z]{3}$/.test(input.toUpperCase())) {
      return input.toUpperCase();
    }

    // Check common airports mapping
    for (const [city, codes] of Object.entries(COMMON_AIRPORTS)) {
      if (normalized.includes(city)) {
        return codes[0]; // Return primary airport
      }
    }

    // Try to match partial city names
    const partialMatches = Object.entries(COMMON_AIRPORTS).filter(([city]) =
>
      city.includes(normalized) || normalized.includes(city)
    );

    if (partialMatches.length > 0) {
      return partialMatches[0][1][0];
    }
```

```
    return null;
  }

  /**
   * Search airports in database
   */
  static async searchAirports(query: string, limit = 10) {
    const searchTerm = `%${query}%`;

    try {
      const results = await db
        .select()
        .from(airports)
        .where(
          or(
            ilike(airports.code, searchTerm),
            ilike(airports.name, searchTerm),
            ilike(airports.city, searchTerm)
          )
        )
        .limit(limit);

      return results;
    } catch (error) {
      console.error('Airport search error:', error);
      return [];
    }
  }

  /**
   * Get airport details by code
   */
  static async getAirportByCode(code: string) {
    try {
      const result = await db
        .select()
```

```
        .from(airports)
        .where(eq(airports.code, code.toUpperCase()))
        .limit(1);

      return result[0] || null;
    } catch (error) {
      console.error('Get airport error:', error);
      return null;
    }
  }

  /**
   * Validate and normalize airport input
   */
  static async validateAirport(input: string): Promise<{ code: string; name?: stri
ng; city?: string } | null> {
    // Try to find code directly
    let code = this.findAirportCode(input);

    if (code) {
      const airportInfo = await this.getAirportByCode(code);
      return {
        code,
        name: airportInfo?.name,
        city: airportInfo?.city,
      };
    }

    // Try database search
    const searchResults = await this.searchAirports(input, 1);
    if (searchResults.length > 0) {
      const airport = searchResults[0];
      return {
        code: airport.code,
        name: airport.name,
        city: airport.city,
```

```
    };
   }

   return null;
  }
}
```

4. Flight Booking Tools Implementation

4.1 Update Chat API Route

Modify `app/(chat)/api/chat/route.ts`:

```
import { z } from 'zod';
import { serpApiService } from '@/lib/services/serpapi';
import { AirportService } from '@/lib/services/airports';
import { db } from '@/lib/db';
import { flightSearches } from '@/lib/db/schema';

// Add these flight booking tools to your existing tools object
const flightBookingTools = {
  searchFlights: {
    description: 'Search for flights between airports using SerpApi Google Fligh
ts',
    parameters: z.object({
      origin: z.string().describe('Origin city or airport code (e.g., "San Francisc
o", "SFO", "New York")'),
      destination: z.string().describe('Destination city or airport code (e.g., "Tok
yo", "NRT", "London")'),
      departureDate: z.string().describe('Departure date in YYYY-MM-DD forma
t'),
      returnDate: z.string().optional().describe('Return date in YYYY-MM-DD for
mat for round-trip flights'),
      passengers: z.number().default(1).describe('Number of adult passenger
s'),
```

```
    travelClass: z.enum(['economy', 'premium_economy', 'business', 'first']).d
efault('economy').describe('Travel class preference'),
  }),
  execute: async ({ origin, destination, departureDate, returnDate, passenger
s, travelClass }) => {
    try {
      // Validate and normalize airport codes
      const originAirport = await AirportService.validateAirport(origin);
      const destinationAirport = await AirportService.validateAirport(destinatio
n);

      if (!originAirport) {
        return {
          error: `Could not find airport for "${origin}". Please provide a valid city
name or 3-letter airport code.`,
          suggestions: 'Try cities like "New York", "Los Angeles", or airport code
s like "JFK", "LAX".'
        };
      }

      if (!destinationAirport) {
        return {
          error: `Could not find airport for "${destination}". Please provide a valid
city name or 3-letter airport code.`,
          suggestions: 'Try cities like "London", "Tokyo", or airport codes like "L
HR", "NRT".'
        };
      }

      // Search flights using SerpApi
      const searchParams = {
        departure_id: originAirport.code,
        arrival_id: destinationAirport.code,
        outbound_date: departureDate,
        return_date: returnDate,
        travel_class: travelClass,
```

```
        adults: passengers,
        type: returnDate ? 1 : 2, // 1 = round-trip, 2 = one-way
      };

      const flightResults = await serpApiService.searchFlights(searchParams);

      // Save search to database
      if (userId && chatId) {
        await db.insert(flightSearches).values({
          userId,
          chatId,
          originCode: originAirport.code,
          destinationCode: destinationAirport.code,
          departureDate: new Date(departureDate),
          returnDate: returnDate ? new Date(returnDate) : null,
          passengers,
          travelClass,
          tripType: returnDate ? 'round-trip' : 'one-way',
          searchResults: flightResults,
        });
      }

      // Format response for AI
      return {
        success: true,
        searchParameters: {
          origin: `${originAirport.city || originAirport.code} (${originAirport.cod
e})`,
          destination: `${destinationAirport.city || destinationAirport.code} (${de
stinationAirport.code})`,
          departureDate,
          returnDate,
          passengers,
          travelClass,
          tripType: returnDate ? 'round-trip' : 'one-way',
        },
```

```
        priceInsights: flightResults.price_insights,
        bestFlights: flightResults.best_flights?.slice(0, 3), // Limit to top 3
        otherFlights: flightResults.other_flights?.slice(0, 5), // Limit to top 5
        totalResults: (flightResults.best_flights?.length || 0) + (flightResults.other
_flights?.length || 0),
        searchUrl: flightResults.search_metadata.google_flights_url,
      };
    } catch (error) {
      console.error('Flight search error:', error);
      return {
        error: 'Sorry, I encountered an error while searching for flights. Please tr
y again or check your search parameters.',
        details: error instanceof Error ? error.message : 'Unknown error'
      };
    }
  }
 },

 getFlightPrice: {
  description: 'Get current price for a specific flight route without full search r
esults',
  parameters: z.object({
    origin: z.string().describe('Origin airport code or city'),
    destination: z.string().describe('Destination airport code or city'),
    departureDate: z.string().describe('Departure date in YYYY-MM-DD forma
t'),
    returnDate: z.string().optional().describe('Return date for round-trip pricin
g'),
  }),
  execute: async ({ origin, destination, departureDate, returnDate }) => {
    try {
      const originAirport = await AirportService.validateAirport(origin);
      const destinationAirport = await AirportService.validateAirport(destinatio
n);

      if (!originAirport || !destinationAirport) {
```

```
        return { error: 'Invalid airport codes provided' };
      }

      const priceInfo = await serpApiService.getFlightPrice(
        originAirport.code,
        destinationAirport.code,
        departureDate,
        returnDate
      );

      return {
        success: true,
        route: `${originAirport.code} → ${destinationAirport.code}`,
        departureDate,
        returnDate,
        lowestPrice: priceInfo.lowest_price,
        priceLevel: priceInfo.price_level, // "low", "typical", "high"
      };
    } catch (error) {
      return {
        error: 'Could not retrieve price information',
        details: error instanceof Error ? error.message : 'Unknown error'
      };
    }
  }
 },

 findAirport: {
  description: 'Find airport codes and information by city or airport name',
  parameters: z.object({
    query: z.string().describe('City name, airport name, or partial search ter
m'),
    limit: z.number().default(5).describe('Maximum number of results to retur
n'),
  }),
  execute: async ({ query, limit }) => {
```

```
      try {
        const airports = await AirportService.searchAirports(query, limit);

        return {
          success: true,
          query,
          results: airports.map(airport => ({
            code: airport.code,
            name: airport.name,
            city: airport.city,
            country: airport.country,
          })),
          totalFound: airports.length,
        };
      } catch (error) {
        return {
          error: 'Could not search airports',
          details: error instanceof Error ? error.message : 'Unknown error'
        };
      }
    }
  },
};

// In your existing chat route, merge flightBookingTools with your existing tools
const tools = {
  ...existingTools, // your current tools (weather, etc.)
  ...flightBookingTools,
};
```

5. UI Components for Flight Display

5.1 Flight Result Card Component

Create `components/ui/flight-card.tsx`:

```
import React from 'react';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/
card';
import { Badge } from '@/components/ui/badge';
import { Clock, Plane, Users, Leaf } from 'lucide-react';
import type { Flight } from '@/lib/services/serpapi';

interface FlightCardProps {
  flight: Flight;
  showPrice?: boolean;
}

export function FlightCard({ flight, showPrice = true }: FlightCardProps) {
  const formatDuration = (minutes: number) => {
    const hours = Math.floor(minutes / 60);
    const mins = minutes % 60;
    return `${hours}h ${mins}m`;
  };

  const formatTime = (timeString: string) => {
    return new Date(`2000-01-01T${timeString}`).toLocaleTimeString('en-US', {
      hour: 'numeric',
      minute: '2-digit',
      hour12: true,
    });
  };

  return (
    <Card className="w-full mb-4 hover:shadow-md transition-shadow">
      <CardHeader className="pb-2">
        <div className="flex justify-between items-start">
          <div className="flex items-center gap-2">
            <img
              src={flight.airline_logo}
              alt={flight.airline}
```

```
            className="w-8 h-8 object-contain"
            onError={(e) => {
              (e.target as HTMLImageElement).style.display = 'none';
            }}
          />
          <div>
            <CardTitle className="text-lg">{flight.airline}</CardTitle>
            <p className="text-sm text-muted-foreground">{flight.flight_numbe
r}</p>
          </div>
        </div>
        {showPrice && flight.price && (
          <div className="text-right">
            <p className="text-2xl font-bold text-primary">${flight.price}</p>
            <p className="text-sm text-muted-foreground">per person</p>
          </div>
        )}
      </div>
    </CardHeader>

    <CardContent>
      <div className="grid grid-cols-3 gap-4 mb-4">
        {/* Departure */}
        <div className="text-center">
          <p className="text-2xl font-semibold">{formatTime(flight.departure_
airport.time)}</p>
          <p className="text-sm font-medium">{flight.departure_airport.id}</p
>
          <p className="text-xs text-muted-foreground">{flight.departure_airp
ort.name}</p>
        </div>

        {/* Duration */}
        <div className="text-center">
          <div className="flex items-center justify-center mb-1">
            <div className="w-8 h-px bg-border"></div>
```

```
            <Plane className="w-4 h-4 mx-2 text-muted-foreground" />
            <div className="w-8 h-px bg-border"></div>
          </div>
          <p className="text-sm font-medium">{formatDuration(flight.duratio
n)}</p>
          {flight.airplane && (
            <p className="text-xs text-muted-foreground">{flight.airplane}</p>
          )}
        </div>

        {/* Arrival */}
        <div className="text-center">
          <p className="text-2xl font-semibold">{formatTime(flight.arrival_airp
ort.time)}</p>
          <p className="text-sm font-medium">{flight.arrival_airport.id}</p>
          <p className="text-xs text-muted-foreground">{flight.arrival_airport.
name}</p>
        </div>
      </div>

      {/* Flight details */}
      <div className="flex flex-wrap gap-2 mb-3">
        <Badge variant="secondary" className="flex items-center gap-1">
          <Users className="w-3 h-3" />
          {flight.travel_class}
        </Badge>

        {flight.legroom && (
          <Badge variant="outline">
            Legroom: {flight.legroom}
          </Badge>
        )}

        {flight.carbon_emissions?.this_flight && (
          <Badge variant="outline" className="flex items-center gap-1">
            <Leaf className="w-3 h-3" />
```

```
            {flight.carbon_emissions.this_flight}g CO₂
          </Badge>
        )}
      </div>

      {/* Extensions */}
      {flight.extensions && flight.extensions.length > 0 && (
        <div className="text-xs text-muted-foreground">
          <p className="mb-1">Amenities:</p>
          <div className="flex flex-wrap gap-1">
            {flight.extensions.slice(0, 3).map((ext, index) => (
              <span key={index} className="inline-block bg-muted px-2 py-1 ro
unded">
                {ext}
              </span>
            ))}
            {flight.extensions.length > 3 && (
              <span className="inline-block bg-muted px-2 py-1 rounded">
                +{flight.extensions.length - 3} more
              </span>
            )}
          </div>
        </div>
      )}
    </CardContent>
  </Card>
 );
}
```

5.2 Flight Search Results Component

Create `components/ui/flight-results.tsx`:

```
import React from 'react';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/
card';
```

```
import { Badge } from '@/components/ui/badge';
import { Button } from '@/components/ui/button';
import { ExternalLink, TrendingUp, TrendingDown, Minus } from 'lucide-react';
import { FlightCard } from './flight-card';
import type { Flight } from '@/lib/services/serpapi';

interface FlightResultsProps {
  searchParameters: {
    origin: string;
    destination: string;
    departureDate: string;
    returnDate?: string;
    passengers: number;
    travelClass: string;
    tripType: string;
  };
  priceInsights?: {
    lowest_price?: number;
    price_level?: string;
    typical_price_range?: number[];
  };
  bestFlights?: Flight[];
  otherFlights?: Flight[];
  totalResults: number;
  searchUrl: string;
}

export function FlightResults({
  searchParameters,
  priceInsights,
  bestFlights,
  otherFlights,
  totalResults,
  searchUrl
}: FlightResultsProps) {
  const getPriceLevelIcon = (level?: string) => {
```

```
  switch (level) {
    case 'low':
      return <TrendingDown className="w-4 h-4 text-green-600" />;
    case 'high':
      return <TrendingUp className="w-4 h-4 text-red-600" />;
    default:
      return <Minus className="w-4 h-4 text-blue-600" />;
  }
};

const getPriceLevelColor = (level?: string) => {
  switch (level) {
    case 'low':
      return 'text-green-600';
    case 'high':
      return 'text-red-600';
    default:
      return 'text-blue-600';
  }
};

return (
  <div className="w-full max-w-4xl mx-auto space-y-4">
    {/* Search Summary */}
    <Card>
      <CardHeader>
        <CardTitle className="flex items-center justify-between">
          <span>Flight Search Results</span>
          <Button asChild variant="outline" size="sm">
            <a href={searchUrl} target="_blank" rel="noopener noreferrer">
              <ExternalLink className="w-4 h-4 mr-2" />
              View on Google Flights
            </a>
          </Button>
        </CardTitle>
      </CardHeader>
```

```
      <CardContent>
        <div className="grid grid-cols-2 md:grid-cols-4 gap-4 text-sm">
          <div>
            <p className="font-medium">Route</p>
            <p className="text-muted-foreground">{searchParameters.origin}
→ {searchParameters.destination}</p>
          </div>
          <div>
            <p className="font-medium">Dates</p>
            <p className="text-muted-foreground">
              {searchParameters.departureDate}
              {searchParameters.returnDate && ` - ${searchParameters.returnDat
e}`}
            </p>
          </div>
          <div>
            <p className="font-medium">Passengers</p>
            <p className="text-muted-foreground">{searchParameters.passen
gers} {searchParameters.travelClass}</p>
          </div>
          <div>
            <p className="font-medium">Results</p>
            <p className="text-muted-foreground">{totalResults} flights found
</p>
          </div>
        </div>
      </CardContent>
    </Card>

    {/* Price Insights */}
    {priceInsights && (
      <Card>
        <CardHeader>
          <CardTitle className="flex items-center gap-2">
            Price Insights
            {getPriceLevelIcon(priceInsights.price_level)}
```

```
          </CardTitle>
        </CardHeader>
        <CardContent>
          <div className="flex items-center gap-4">
            {priceInsights.lowest_price && (
              <div>
                <p className="text-2xl font-bold">${priceInsights.lowest_price}
</p>
                <p className="text-sm text-muted-foreground">Lowest price fou
nd</p>
              </div>
            )}
            {priceInsights.price_level && (
              <Badge variant="outline" className={`${getPriceLevelColor(pricel
nsights.price_level)} capitalize`}>
                {priceInsights.price_level} price level
              </Badge>
            )}
            {priceInsights.typical_price_range && (
              <div>
                <p className="text-sm text-muted-foreground">
                  Typical range: ${priceInsights.typical_price_range[0]} - ${priceIns
ights.typical_price_range[1]}
                </p>
              </div>
            )}
          </div>
        </CardContent>
      </Card>
    )}

    {/* Best Flights */}
    {bestFlights && bestFlights.length > 0 && (
      <div>
        <h3 className="text-lg font-semibold mb-3">Best Flights</h3>
        {bestFlights.map((flight, index) => (
```

```
          <FlightCard key={`best-${index}`} flight={flight} />
        ))}
      </div>
    )}

    {/* Other Flights */}
    {otherFlights && otherFlights.length > 0 && (
      <div>
        <h3 className="text-lg font-semibold mb-3">Other Options</h3>
        {otherFlights.map((flight, index) => (
          <FlightCard key={`other-${index}`} flight={flight} />
        ))}
      </div>
    )}
  </div>
 );
}
```

6. Artifact Integration

6.1 Update Artifact System

Add to your existing artifact types in `lib/artifacts.ts`:

```
export const artifactTypes = {
 // ... existing types
 'flight-results': {
  component: 'FlightResults',
  title: 'Flight Search Results',
  description: 'Interactive flight search results with pricing and booking optio
ns',
 },
};
```

6.2 Create Flight Results Artifact Component

Create `components/artifacts/flight-results-artifact.tsx`:

```
import React from 'react';
import { FlightResults } from '@/components/ui/flight-results';

interface FlightResultsArtifactProps {
  data: {
    searchParameters: any;
    priceInsights?: any;
    bestFlights?: any[];
    otherFlights?: any[];
    totalResults: number;
    searchUrl: string;
  };
}

export function FlightResultsArtifact({ data }: FlightResultsArtifactProps) {
  return (
    <div className="w-full">
      <FlightResults {...data} />
    </div>
  );
}
```

7. Database Seeding

7.1 Airport Data Seeding

Create `scripts/seed-airports.ts`:

```
import { db } from '@/lib/db';
import { airports } from '@/lib/db/schema';

const MAJOR_AIRPORTS = [
  // North America
```

```
  { code: 'JFK', name: 'John F. Kennedy International Airport', city: 'New York', country: 'United States', timezone: 'America/New_York' },
  { code: 'LAX', name: 'Los Angeles International Airport', city: 'Los Angeles', country: 'United States', timezone: 'America/Los_Angeles' },
  { code: 'ORD', name: "O'Hare International Airport", city: 'Chicago', country: 'United States', timezone: 'America/Chicago' },
  { code: 'DFW', name: 'Dallas/Fort Worth International Airport', city: 'Dallas', country: 'United States', timezone: 'America/Chicago' },
  { code: 'DEN', name: 'Denver International Airport', city: 'Denver', country: 'United States', timezone: 'America/Denver' },
  { code: 'SFO', name: 'San Francisco International Airport', city: 'San Francisco', country: 'United States', timezone: 'America/Los_Angeles' },
  { code: 'SEA', name: 'Seattle-Tacoma International Airport', city: 'Seattle', country: 'United States', timezone: 'America/Los_Angeles' },
  { code: 'MIA', name: 'Miami International Airport', city: 'Miami', country: 'United States', timezone: 'America/New_York' },
  { code: 'BOS', name: 'Logan International Airport', city: 'Boston', country: 'United States', timezone: 'America/New_York' },
  { code: 'ATL', name: 'Hartsfield-Jackson Atlanta International Airport', city: 'Atlanta', country: 'United States', timezone: 'America/New_York' },

  // Europe
  { code: 'LHR', name: 'Heathrow Airport', city: 'London', country: 'United Kingdom', timezone: 'Europe/London' },
  { code: 'CDG', name: 'Charles de Gaulle Airport', city: 'Paris', country: 'France', timezone: 'Europe/Paris' },
  { code: 'FRA', name: 'Frankfurt Airport', city: 'Frankfurt', country: 'Germany', timezone: 'Europe/Berlin' },
  { code: 'AMS', name: 'Amsterdam Airport Schiphol', city: 'Amsterdam', country: 'Netherlands', timezone: 'Europe/Amsterdam' },
  { code: 'MAD', name: 'Adolfo Suárez Madrid–Barajas Airport', city: 'Madrid', country: 'Spain', timezone: 'Europe/Madrid' },
  { code: 'FCO', name: 'Leonardo da Vinci–Fiumicino Airport', city: 'Rome', country: 'Italy', timezone: 'Europe/Rome' },
  { code: 'ZUR', name: 'Zurich Airport', city: 'Zurich', country: 'Switzerland', timezone: 'Europe/Zurich' },
```

```
  { code: 'VIE', name: 'Vienna International Airport', city: 'Vienna', country: 'Aus
tria', timezone: 'Europe/Vienna' },

  // Asia
  { code: 'NRT', name: 'Narita International Airport', city: 'Tokyo', country: 'Jap
an', timezone: 'Asia/Tokyo' },
  { code: 'HND', name: 'Haneda Airport', city: 'Tokyo', country: 'Japan', timezon
e: 'Asia/Tokyo' },
  { code: 'PEK', name: 'Beijing Capital International Airport', city: 'Beijing', coun
try: 'China', timezone: 'Asia/Shanghai' },
  { code: 'PVG', name: 'Shanghai Pudong International Airport', city: 'Shangha
i', country: 'China', timezone: 'Asia/Shanghai' },
  { code: 'HKG', name: 'Hong Kong International Airport', city: 'Hong Kong', co
untry: 'Hong Kong', timezone: 'Asia/Hong_Kong' },
  { code: 'SIN', name: 'Singapore Changi Airport', city: 'Singapore', country: 'Si
ngapore', timezone: 'Asia/Singapore' },
  { code: 'ICN', name: 'Incheon International Airport', city: 'Seoul', country: 'So
uth Korea', timezone: 'Asia/Seoul' },
  { code: 'BOM', name: 'Chhatrapati Shivaji Maharaj International Airport', city:
'Mumbai', country: 'India', timezone: 'Asia/Kolkata' },
  { code: 'DEL', name: 'Indira Gandhi International Airport', city: 'Delhi', countr
y: 'India', timezone: 'Asia/Kolkata' },
  { code: 'DXB', name: 'Dubai International Airport', city: 'Dubai', country: 'Unit
ed Arab Emirates', timezone: 'Asia/Dubai' },
];

export async function seedAirports() {
  try {
    console.log('Seeding airports...');

    for (const airport of MAJOR_AIRPORTS) {
      await db.insert(airports).values(airport).onConflictDoNothing();
    }

    console.log(`Seeded ${MAJOR_AIRPORTS.length} airports successfully`);
  } catch (error) {
```

```
    console.error('Error seeding airports:', error);
    throw error;
  }
}

// Run if this file is executed directly
if (require.main === module) {
  seedAirports()
    .then(() => process.exit(0))
    .catch((error) => {
      console.error(error);
      process.exit(1);
    });
}
```

8. Testing Implementation

8.1 Test Flight Search Function

Create `tests/flight-search.test.ts`:

```
import { describe, it, expect, beforeEach } from 'vitest';
import { serpApiService } from '@/lib/services/serpapi';
import { AirportService } from '@/lib/services/airports';

describe('Flight Search Integration', () => {
  beforeEach(() => {
    // Mock environment variables
    process.env.SERPAPI_KEY = 'test_key';
  });

  it('should find airport codes from city names', async () => {
    const result = AirportService.findAirportCode('New York');
    expect(result).toBe('JFK');
```

```
    const result2 = AirportService.findAirportCode('san francisco');
    expect(result2).toBe('SFO');
  });

  it('should validate existing airport codes', async () => {
    const result = AirportService.findAirportCode('LAX');
    expect(result).toBe('LAX');
  });

  // Add more tests as needed
});
```

9. Deployment Checklist

9.1 Environment Variables

Ensure these are set in production:

```
SERPAPI_KEY=your_production_serpapi_key
SERPAPI_BASE_URL=https://serpapi.com/search.json
```

9.2 Database Migrations

Run these commands in order:

```
# Generate new migrations
npm run db:generate

# Apply migrations
npm run db:migrate

# Seed airports data
npm run seed:airports
```

9.3 Rate Limiting Considerations

- SerpApi free tier: 100 searches/month
- Paid tier: $50/month for 5000 searches
- Implement caching for repeated searches
- Consider adding user rate limiting

10. Usage Examples

10.1 Example User Interactions

```
User: "Find flights from New York to London on December 25th"
AI: [Calls searchFlights tool, returns flight results in artifact]

User: "What about business class?"
AI: [Calls searchFlights with travel_class: 'business']

User: "How much would a round trip cost returning January 5th?"
AI: [Calls searchFlights with return date]

User: "Set a price alert for under $500"
AI: [Future feature - price monitoring]
```

10.2 Error Handling Examples

```
User: "Flights from XYZ to ABC"
AI: "I couldn't find airports for those codes. Did you mean a specific city?"

User: "Flights for tomorrow" (without specifying route)
AI: "I need both departure and arrival cities to search for flights."
```

11. Future Enhancements

11.1 Phase 2 Features

- Price monitoring and alerts

- Multi-city trip planning
- Hotel and car rental integration
- Booking URL generation
- Calendar view for flexible dates

11.2 Performance Optimizations

- Result caching for popular routes
- Background price updates
- Concurrent API calls for round trips
- Image lazy loading for airline logos

Implementation Priority

1. **Phase 1**: Basic flight search (searchFlights tool)
2. **Phase 2**: UI components and artifacts
3. **Phase 3**: Database integration and search history
4. **Phase 4**: Price monitoring and alerts
5. **Phase 5**: Advanced features and optimizations

This implementation guide provides a complete foundation for converting your chatbot into a flight booking agent using SerpApi. Start with Phase 1 for immediate functionality, then incrementally add features based on user feedback and requirements.