Restaurant Planner - System Design Document (v2)

System Overview

A collaborative restaurant planning application that uses AI to find and book restaurants based on group preferences. The system collects individual dietary preferences and location requirements, learns from user feedback through restaurant dislikes and post-event reviews, then autonomously recommends and books suitable venues.

User Flow

Phase 1: User Onboarding

Trigger: First-time visitor lands on the website

Process: 1. User completes a brief onboarding quiz capturing: - Dietary restrictions/ preferences - Alcohol preference (alcoholic/non-alcoholic) 2. User provides phone number as unique identifier 3. System stores preferences linked to phone number

Phase 2: Event Creation

Trigger: User (now "Organizer") wants to create a dining event

Process: 1. Organizer specifies: - **Location:** City name or city district (can be broad or specific) - **Description:** Type of dining occasion (e.g., "birthday dinner", "business lunch") - **Expected attendees:** Number of people (optional) - **Preferred date/time:** One or more time slots with dates 2. System generates: - Unique event ID - Shareable invitation link 3. Organizer can share link with potential attendees

Phase 3: Guest Response

Trigger: Invitee receives and clicks invitation link

Process: 1. Invitee is redirected to response form 2. Invitee either: - Completes mini-quiz if new user (same as Phase 1) - Uses existing preferences if returning user 3. Invitee provides event-specific information: - Attendance confirmation (consent) - Location preference override (optional) - Event-specific dietary restrictions (optional) 4. Response is recorded and linked to event

Phase 4: AI Restaurant Selection

Trigger: Number of confirmed attendees matches organizer's expected count OR organizer manually triggers search

Process: 1. AI agent retrieves from database: - All confirmed attendee preferences - Event parameters (location, time, date, occasion type) - **Restaurant blacklist:** All restaurants disliked by any attendee 2. Agent analyzes collective requirements: - Identifies common location preferences - Aggregates all dietary restrictions - Considers alcohol preferences - Factors in occasion type - **Excludes blacklisted restaurants** 3. Agent generates ranked list of top 5 restaurant recommendations with justification

Restaurant Blacklist Logic: - If ANY attendee has marked a restaurant as "disliked", it is excluded from recommendations - Dislikes can be permanent (user-level) or event-specific - AI logs which restaurants were excluded and why

Phase 5: Restaurant Booking

Trigger: AI completes restaurant selection

Process: 1. Agent contacts restaurants (via phone/API) in ranked order 2. For each restaurant: - Checks availability for specified date/time/party size - If available: proceeds to booking - If unavailable: moves to next option 3. First available restaurant is booked

Phase 6: Confirmation & Calendar Integration

Trigger: Successful restaurant booking

Process: 1. System sends Google Calendar invitations to all confirmed attendees 2. Invitation includes: - Restaurant name and address - Booking time and date - Number of attendees - Special notes (dietary accommodations, etc.)

Phase 7: Post-Event Review

Trigger: Event date has passed (24-48 hours after scheduled time)

Process: 1. System sends push notification/email to all attendees 2. Each attendee can provide: - **Overall rating:** 1-5 stars - **Category ratings:** - Food quality (1-5 stars) - Service quality (1-5 stars) - Atmosphere (1-5 stars) - Value for money (1-5 stars) - **Written remarks:** Free-text feedback - **Dislike flag:** Mark restaurant to exclude from future recommendations 3. Feedback is stored and used to improve future AI recommendations 4. Aggregate ratings are calculated for the restaurant

Flexible Grading Scheme:

```
Overall Experience: 1-5 stars (required)
├── Food Quality: 1-5 stars (optional)
├── Service: 1-5 stars (optional)
├── Atmosphere: 1-5 stars (optional)
└── Value: 1-5 stars (optional)

Additional Feedback:
├── Written remarks (optional, max 500 chars)
├── Would recommend: Yes/No/Maybe (optional)
└── Add to blacklist: Yes/No (explicit action)
```

Database Schema (Firebase Firestore)

Collection: `users`

Stores user profile and preferences.

Document ID: `{phone_number}` (e.g., "+31612345678")

Fields:

```
{
  phone_number: string,          // Primary identifier
  email: string,                 // For calendar invitations
  dietary_restrictions: array,   // e.g., ["vegetarian", "gluten-free"]
  alcohol_preference: string,    // "alcoholic" | "non-alcoholic" | "no-preference"

  // Notification preferences
  push_notifications_enabled: boolean,
  email_notifications_enabled: boolean,

  created_at: timestamp,
  updated_at: timestamp
}
```

Collection: `events`

Stores event information created by organizers.

Document ID: Auto-generated unique ID (e.g., "evt_abc123xyz")

Fields:

```
{
  event_id: string,              // Unique event identifier
  organizer_phone: string,       // Reference to users collection
  organizer_email: string,       // For notifications

  // Event details
  location: string,              // "Amsterdam Centre" or "Amsterdam"
  occasion_description: string,  // "Birthday dinner", "Team outing"
  expected_attendee_count: number | null,  // Optional

  // Timing
  preferred_date: timestamp,
  preferred_time_slots: array,   // e.g., ["18:00-20:00", "20:00-22:00"]

  // Status
  status: string,                // "created" | "collecting_responses" | "ready_for_booking" | "booked" | "c

  // Post-event
  reviews_requested: boolean,    // Has review notification been sent
  reviews_requested_at: timestamp | null,

  // Metadata
  created_at: timestamp,
  updated_at: timestamp,
  invitation_link: string        // Full shareable URL
}
```

Collection: `event_responses`

Stores individual responses from invitees for each event.

Document ID: Auto-generated

Fields:

```
{
  event_id: string,              // Reference to events collection
  respondent_phone: string,      // Reference to users collection
  respondent_email: string,

  // Response data
  attendance_confirmed: boolean,

  // Event-specific preferences (override user defaults)
  location_preference_override: string | null,  // If different from event location
  event_specific_dietary_notes: string | null,  // Additional restrictions for this event

  // Metadata
  responded_at: timestamp,
  updated_at: timestamp
}
```

Indexes needed: - Composite index on `(event_id, attendance_confirmed)`

Collection: `restaurant_dislikes`

Stores user dislikes for restaurants (blacklist).

Document ID: Auto-generated

Fields:

```
{
  user_phone: string,            // Reference to users collection
  restaurant_name: string,       // Name of disliked restaurant
  restaurant_address: string,    // Full address for unique identification

  // Dislike context
  dislike_type: string,          // "permanent" | "event-specific"
  event_id: string | null,       // If event-specific, which event

  // Reason (optional)
  reason: string | null,         // "poor_food", "bad_service", "allergy_concern", "other"
  notes: string | null,          // Additional context

  // Metadata
  created_at: timestamp,
  is_active: boolean             // Can be deactivated without deleting
}
```

Indexes needed: - Composite index on `(user_phone, is_active)` - Composite index on `(restaurant_name, is_active)`

Collection: `restaurant_recommendations`

Stores AI-generated restaurant recommendations for each event.

Document ID: Auto-generated

Fields:

```
{
  event_id: string,              // Reference to events collection

  recommendations: array,        // Ordered list (top 5)
  // Each recommendation object:
  // {
  //   rank: number,
  //   restaurant_name: string,
  //   address: string,
  //   phone: string,
  //   cuisine_type: string,
  //   reasoning: string,         // Why this restaurant was selected
  //   accommodates_requirements: object  // Which dietary needs are met
  // }

  // Analysis summary
  collective_dietary_restrictions: array,
  alcohol_availability_required: boolean,
  location_consensus: string,

  // Blacklist information
  excluded_restaurants: array,   // Restaurants excluded due to dislikes
  // Each excluded restaurant:
  // {
  //   restaurant_name: string,
  //   excluded_by: array,        // Phone numbers of users who disliked it
  //   dislike_reasons: array     // Reasons provided
  // }

  // AI metadata
  generated_at: timestamp,
  ai_model_used: string,
  confidence_score: number       // 0-1 scale
}
```

Collection: `bookings`

Stores final booking information.

Document ID: Auto-generated

Fields:

```
{
  event_id: string,               // Reference to events collection
  recommendation_id: string,      // Which recommendation was booked

  // Restaurant details
  restaurant_name: string,
  restaurant_address: string,
  restaurant_phone: string,
  restaurant_cuisine_type: string,

  // Booking details
  booking_date: timestamp,
  booking_time: string,           // "19:00"
  party_size: number,
  booking_confirmation_number: string | null,

  // Status
  booking_status: string,         // "pending" | "confirmed" | "completed" | "cancelled"

  // Calendar integration
  calendar_invites_sent: boolean,
  calendar_event_ids: array,      // Google Calendar event IDs for each attendee

  // Metadata
  booked_at: timestamp,
  updated_at: timestamp,
  completed_at: timestamp | null // When event actually happened
}
```

Indexes needed: - Index on `booking_status` - Index on `booking_date`

Collection: `post_event_reviews`

Stores reviews submitted by attendees after the event.

Document ID: Auto-generated

Fields:

```
{
  event_id: string,              // Reference to events collection
  booking_id: string,            // Reference to bookings collection
  reviewer_phone: string,        // Reference to users collection

  // Restaurant being reviewed
  restaurant_name: string,
  restaurant_address: string,

  // Ratings (1-5 scale)
  overall_rating: number,        // Required (1-5)
  food_quality_rating: number | null,   // Optional (1-5)
  service_rating: number | null,        // Optional (1-5)
  atmosphere_rating: number | null,     // Optional (1-5)
  value_rating: number | null,          // Optional (1-5)

  // Qualitative feedback
  would_recommend: string | null,       // "yes" | "no" | "maybe"
  written_remarks: string | null,       // Max 500 characters

  // Dislike action
  added_to_blacklist: boolean,   // Did user dislike this restaurant

  // Metadata
  submitted_at: timestamp,
  updated_at: timestamp
}
```

Indexes needed: - Composite index on `(event_id, reviewer_phone)` - Index on `restaurant_name` (for aggregate statistics) - Index on `overall_rating`

Collection: `restaurant_aggregate_ratings`

Stores aggregate ratings for restaurants across all events.

Document ID: `{restaurant_name}_{normalized_address}` (unique identifier per restaurant)

Fields:

```
{
  restaurant_name: string,
  restaurant_address: string,

  // Aggregate metrics
  total_reviews: number,
  average_overall_rating: number,      // Calculated average
  average_food_rating: number,
  average_service_rating: number,
  average_atmosphere_rating: number,
  average_value_rating: number,

  // Distribution
  rating_distribution: object,   // { "1": 0, "2": 1, "3": 5, "4": 10, "5": 3 }

  // Recommendation metrics
  would_recommend_yes: number,
  would_recommend_no: number,
  would_recommend_maybe: number,

  // Blacklist information
  total_blacklists: number,      // How many users have blacklisted this
  blacklist_percentage: number,  // Percentage of reviewers who blacklisted

  // Metadata
  first_review_date: timestamp,
  last_review_date: timestamp,
  last_updated: timestamp
}
```

Indexes needed: - Index on `average_overall_rating` - Index on `total_reviews`

Collection: `ai_agent_logs`

Stores logs of AI agent actions for debugging and auditing.

Document ID: Auto-generated

Fields:

```
{
  event_id: string,
  action_type: string,           // "recommendation_generation" | "restaurant_call" | "booking_attempt"

  // Action details
  input_data: object,            // What data the agent received
  output_data: object,           // What the agent produced

  // Call details (for restaurant calls)
  restaurant_contacted: string | null,
  call_transcript: string | null,
  call_outcome: string | null,   // "available" | "unavailable" | "no_answer"

  // Blacklist tracking
  blacklisted_restaurants_excluded: number,

  // Status
  success: boolean,
  error_message: string | null,

  // Metadata
  timestamp: timestamp,
  duration_ms: number
}
```

Key Relationships

```
users (1) ─────< (many) event_responses
users (1) ─────< (many) restaurant_dislikes
users (1) ─────< (many) post_event_reviews

events (1) ────< (many) event_responses
events (1) ────< (1) restaurant_recommendations
events (1) ────< (1) bookings
events (1) ────< (many) post_event_reviews

bookings (1) ──< (many) post_event_reviews

restaurant_dislikes (many) ───> (influences) restaurant_recommendations
post_event_reviews (many) ────> (aggregates to) restaurant_aggregate_ratings
```

API Endpoints (Flask Backend)

User Management

- `POST /api/users/onboarding` - Create/update user preferences
- `GET /api/users/{phone_number}` - Retrieve user profile
- `PATCH /api/users/{phone_number}/preferences` - Update notification preferences

Event Management

- `POST /api/events` - Create new event
- `GET /api/events/{event_id}` - Retrieve event details
- `PATCH /api/events/{event_id}` - Update event
- `DELETE /api/events/{event_id}` - Cancel event

Event Responses

- `POST /api/events/{event_id}/responses` - Submit invitee response
- `GET /api/events/{event_id}/responses` - Get all responses for event

Restaurant Dislikes

- `POST /api/users/{phone_number}/dislikes` - Add restaurant to blacklist
- `GET /api/users/{phone_number}/dislikes` - Get user's blacklisted restaurants
- `DELETE /api/users/{phone_number}/dislikes/{dislike_id}` - Remove from blacklist
- `PATCH /api/users/{phone_number}/dislikes/{dislike_id}` - Update dislike status

AI Agent

- `POST /api/events/{event_id}/generate-recommendations` - Trigger AI restaurant selection
- `GET /api/events/{event_id}/recommendations` - Retrieve recommendations

Booking

- `POST /api/events/{event_id}/book` - Execute restaurant booking
- `GET /api/bookings/{booking_id}` - Retrieve booking details
- `PATCH /api/bookings/{booking_id}/complete` - Mark booking as completed

Post-Event Reviews

- `POST /api/events/{event_id}/request-reviews` - Trigger review notifications
- `POST /api/events/{event_id}/reviews` - Submit a review

- `GET /api/events/{event_id}/reviews` - Get all reviews for an event
- `GET /api/restaurants/{restaurant_id}/reviews` - Get all reviews for a restaurant
- `GET /api/restaurants/{restaurant_id}/aggregate-rating` - Get aggregate ratings

Restaurant Dislike/Blacklist System

How Dislikes Work

Adding a Dislike: 1. User can add a restaurant to blacklist from two places: - **Post-event review:** After rating, user can check "Don't show me this restaurant again" - **Manual addition:** User can proactively add restaurants to their blacklist

Dislike Types: - **Permanent:** Restaurant will never be recommended to this user - **Event-specific:** Only excluded for a particular event (rarely used)

When Dislikes Are Applied: - During Phase 4 (AI Restaurant Selection), the AI queries all dislikes for all attendees - ANY restaurant disliked by ANY attendee is automatically excluded - The AI logs which restaurants were excluded and why - If too many restaurants are blacklisted (>50% of viable options), system warns organizer

Dislike Reasons:

```
- poor_food: "Food quality was unsatisfactory"
- bad_service: "Service was poor"
- allergy_concern: "Had allergic reaction or cross-contamination"
- atmosphere: "Didn't like the atmosphere"
- value: "Not worth the price"
- personal: "Personal preference"
- other: "Other reason" (with optional notes)
```

Post-Event Review System

Review Request Flow

Timing: - Reviews are requested 24-48 hours after the scheduled event time - Configurable delay in system settings

Notification Method: 1. Push notification (if enabled) 2. Email (if push fails or disabled) 3. In-app banner (when user next visits)

Review Form Structure:

```
┌─────────────────────────────────────────┐
│ How was [Restaurant Name]?              │
├─────────────────────────────────────────┤
│                                         │
│ Overall Experience: ⭐⭐⭐⭐⭐ (required)  │
│                                         │
│ ▼ Rate specific aspects (optional)     │
│   Food Quality: ⭐⭐⭐⭐☆                 │
│   Service: ⭐⭐⭐⭐⭐                      │
│   Atmosphere: ⭐⭐⭐☆☆                  │
│   Value: ⭐⭐⭐⭐☆                       │
│                                         │
│ Would you recommend this restaurant?    │
│ ○ Yes  ○ Maybe  ○ No                   │
│                                         │
│ ▼ Any additional remarks? (optional)   │
│ [Text area - 500 char max]             │
│                                         │
│ ☐ Don't show me this restaurant again  │
│                                         │
│ [Submit Review]  [Skip]                │
└─────────────────────────────────────────┘
```

Using Review Data

For AI Recommendations: - Restaurants with average rating < 3.0 are deprioritized - Restaurants with >20% blacklist rate are flagged for review - Positive reviews boost restaurant's ranking in future recommendations

For Users: - Users can view their past reviews - Users can see aggregate ratings when viewing recommendations

---

Open Questions & Decisions Needed

1. **Mini-quiz for invitees:** Should invitees who already have profiles still see their preferences and have the option to modify them for this specific event?
2. **Recommendation:** Yes, show existing preferences with option to add event-specific notes
3. **Consent threshold:** Should booking proceed automatically when expected count is reached, or should organizer manually approve?
4. **Recommendation:** Add `auto_book: boolean` field to events, let organizer decide
5. **Restaurant calling:** Will this be actual phone calls (using service like Twilio) or API integrations with restaurant booking platforms?

6. **Recommendation:** Start with API integrations (OpenTable, TheFork) as fallback to manual organizer booking if unavailable
7. **Time slot selection:** Should AI choose the best time slot from options, or should organizer pre-select one?
8. **Recommendation:** Let organizer provide ranked preferences, AI selects based on restaurant availability
9. **Email collection:** When do we collect email addresses?
10. **Recommendation:** During onboarding AND when responding to event (some people might use different emails)
11. **Review reminder frequency:** How many times should we remind users to submit reviews?
12. **Recommendation:** Maximum 2 reminders: initial notification + 1 follow-up after 7 days if not completed
13. **Blacklist threshold:** Should we warn users/organizers if too many restaurants are blacklisted?
14. **Recommendation:** Yes, warn if >50% of potential restaurants are excluded, suggest expanding location or criteria
15. **Anonymous reviews:** Should reviews be anonymous to other users or visible with name?
16. **Recommendation:** Anonymous to public, but event organizer can see who submitted what

---

Tech Stack Implementation Notes

Frontend (Vite + React)

Key Libraries: - React Router - for navigation and invitation link handling - React Hook Form - for quiz and event creation forms - Axios - for API calls - React Query - for data fetching and caching - React Toastify - for notifications - Star Rating Component - for review interface

Key Pages: - `/` - Home/Landing - `/onboarding` - User quiz - `/events/create` - Event creation form - `/events/:eventId/respond` - Invitation response page - `/events/:eventId/review` - Post-event review page - `/profile` - User profile and blacklist management

Backend (Flask)

Key Libraries: - Flask-CORS - for cross-origin requests - Firebase Admin SDK - for Firestore access - LangChain - for AI agent - Google Calendar API - for calendar integration - Twilio/OpenTable API - for restaurant booking

Key Modules: - `routes/` - API endpoint handlers - `services/ai_agent.py` - LangChain AI logic - `services/booking.py` - Restaurant booking logic - `services/notification.py` - Email/push notifications - `services/calendar.py` - Google Calendar integration

Database (Firebase Firestore)

Configuration: - Set up composite indexes as specified in schema - Enable security rules for data protection - Set up backup schedule

AI Agent (LangChain)

Components: - Restaurant recommendation chain - Preference analysis chain - Booking conversation chain - Review sentiment analysis (optional)

Data Sources: - User preferences from Firestore - Restaurant data from APIs (Yelp, Google Places) - Historical review data