# Lake Ride Pros Chatbot - Booking Integration Upgrade

## Project Context

You are working on a Firebase-based transportation booking application (Lake Ride Pros) with an existing AI chatbot. The chatbot currently uses GPT-4o-mini to answer questions from a knowledge base, but needs to be upgraded to intelligently collect booking details and send them to the business owners.

## Current Architecture

Firebase Functions Structure:
- `functions/index.js` - Main exports file using `attach()` pattern
- `functions/chatbotQuery.js` - Current chatbot implementation (OpenAI integration)
- Existing email via Gmail API (see `sendShuttleTicketEmail`, `notifyQueueOnCreate`)
- Existing SMS via `smsOnCreateV2` function
- Settings stored in Firestore: `appSettings/chatbot` and `appSettings/aiContentGenerator`

Current Chatbot Flow:
1. External website loads widget via `chatbotConfig` endpoint
2. User chats with bot via `chatbotQuery` endpoint (no auth required)
3. Bot uses GPT-4o-mini + knowledge base to answer questions
4. Manual escalation to Facebook Messenger via button click

## Objective

Upgrade the chatbot to:
1. **Intelligently detect booking intent** and collect required information conversationally
2. **Extract structured booking data**: name, email, phone, pickup, dropoff, date, time, passengers, special requests
3. **Send structured email** to `owners@lakeridepros.com` using existing Gmail API infrastructure
4. **Send SMS confirmation** to customer using existing SMS infrastructure
5. **Store booking request** in Firestore for tracking and analytics
6. **Add smart guardrails** to prevent GPT-4o-mini hallucinations about pricing/availability

## Critical Requirements

### Cost Management
- **KEEP GPT-4o-mini** (don't upgrade to gpt-4o unless necessary)
- Add validation layers to catch hallucinations
- Implement confidence scoring for answers
- Target: <$0.002 per conversation

### Safety Guardrails
- Bot must NEVER discuss specific pricing (escalate to human)
- Bot must NEVER guarantee availability (escalate to human)
- Bot must NEVER make commitments about services
- If unsure, always escalate to Messenger

### Data Collection
Required fields:
- `customer_name` (full name)
- `customer_email` (validated format)
- `customer_phone` (US format preferred)
- `pickup_location` (specific address when possible)
- `dropoff_location` (specific address when possible)
- `trip_date` (YYYY-MM-DD)
- `trip_time` (HH:MM 24-hour)
- `passenger_count` (number)
- `trip_type` (one-way, round-trip, hourly, event, airport) **REQUIRED**

Optional but helpful:
- `special_requests` (car seats, wheelchair, luggage, etc.)

### User Experience
- Conversational, NOT a boring form
- Ask for details naturally as conversation flows
- Confirm all details before submitting
- Show clear success message with booking reference ID
- Respond within 15 minutes promise to customer

## Implementation Tasks

### Task 1: Add GPT Function Calling to chatbotQuery.js

Location: `functions/chatbotQuery.js`

What to do:
1. Add OpenAI function calling tool definition for `submit_booking_request`
2. Update `queryOpenAI()` to support tools parameter
3. Add response handling for tool calls
4. Implement validation for required booking fields

Function calling schema:
```javascript
const BOOKING_TOOL = {
  type: "function",
  function: {
    name: "submit_booking_request",
    description: "Submit complete booking request after collecting all required customer information",
    parameters: {
      type: "object",
      properties: {
        customer_name: { type: "string" },
        customer_email: { type: "string" },
        customer_phone: { type: "string" },
        pickup_location: { type: "string" },
        dropoff_location: { type: "string" },
        trip_date: { type: "string" },
        trip_time: { type: "string" },
        passenger_count: { type: "number" },
        special_requests: { type: "string" },
        trip_type: { type: "string", enum: ["one-way", "round-trip", "hourly", "event", "airport"] }
      },
      required: [
        "customer_name",
        "customer_email",
        "customer_phone",
        "pickup_location",
        "dropoff_location",
        "trip_date",
        "trip_time",
        "passenger_count",
        "trip_type"
      ]
    }
  }
};
```

Updated system prompt:
```javascript
const systemPrompt = `You are Johnny, Lake Ride Pros' friendly booking assistant.

YOUR GOAL: Collect booking details naturally through conversation, then submit via the submit_booking_request function.

REQUIRED INFORMATION:
1. Customer name (first and last)
2. Email address
3. Phone number
4. Pickup location (get specific address if possible)
5. Dropoff location (get specific address if possible)
6. Date of trip
7. Time of pickup
8. Number of passengers
9. Trip type (one-way, round-trip, hourly, event, or airport)

CONVERSATION STYLE:
- Be warm and conversational, NOT robotic
- Don't ask for everything at once - flow naturally
- If they give multiple details, acknowledge them all
```

- Ask clarifying questions if location/time is vague
- Confirm details before submitting: "Let me make sure I have everything correct..."

CRITICAL RULES - ALWAYS FOLLOW:
- NEVER discuss specific pricing (say: "Our team will text you exact pricing within 15 minutes")
- NEVER guarantee availability (say: "We'll confirm availability right away")
- NEVER make up information about services
- If uncertain about ANYTHING, say: "Let me connect you with our team on Messenger for accurate details"

When you have ALL required information and customer confirms, call submit_booking_request.

${context}`;

Update queryOpenAI function:
```javascript
async function queryOpenAI(settings, messages, tools = null) {
  const body = {
    model: settings.model || "gpt-4o-mini",
    messages,
    temperature: 0.3, // Lower temp for booking = less creative = fewer hallucinations
    max_tokens: 500,
  };

  if (tools) {
    body.tools = tools;
    body.tool_choice = "auto";
  }

  const response = await fetch("https://api.openai.com/v1/chat/completions", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${settings.apiKey}`,
    },
    body: JSON.stringify(body),
  });

  if (!response.ok) {
    const errorData = await response.json().catch(() => ({}));
    throw new Error(errorData.error?.message || `API error: ${response.status}`);
  }

  return await response.json();
}
```

### Task 2: Create Email Handler for Booking Requests

Location: Create new file `functions/sendBookingRequestEmail.js`

What to do:
1. Create function that uses existing Gmail API setup (same pattern as `sendShuttleTicketEmail`)
2. Format booking details in HTML email
3. Include quick action buttons (call, text, create in Moovs)
4. Send to `owners@lakeridepros.com`

Reference existing Gmail implementation:
Look at `functions/sendShuttleTicketEmail.js` for the Gmail API pattern you're already using. Follow the same authentication and sending approach.

Email template structure:
```javascript
const emailHtml = `
<!DOCTYPE html>
<html>
<head>
  <style>
    body { font-family: Arial, sans-serif; line-height: 1.6; color: #333; }
    .container { max-width: 600px; margin: 0 auto; padding: 20px; }
    .header { background: #4CAF50; color: white; padding: 20px; text-align: center; }
    .urgent { background: #fff3cd; border-left: 4px solid #ffc107; padding: 15px; margin: 15px 0; }
    .field { margin: 10px 0; padding: 8px; background: #f9f9f9; }
    .label { font-weight: bold; color: #555; }
    .button {
      display: inline-block;
      padding: 12px 24px;
      margin: 5px;
      background: #4CAF50;
      color: white;
      text-decoration: none;
      border-radius: 4px;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <h2>🚗 New Booking Request from AI Chatbot</h2>
    </div>

    <div class="urgent">
      <strong>⏰ RESPOND WITHIN 15 MINUTES</strong><br>
      Customer expects quick reply via text/email
    </div>

    <h3>Customer Information</h3>
    <div class="field"><span class="label">Name:</span> ${bookingData.customer_name}</div>
    <div class="field"><span class="label">Phone:</span> <a href="tel:${bookingData.customer_phone}">${bookingData.customer_phone}</a></div>
    <div class="field"><span class="label">Email:</span> <a href="mailto:${bookingData.customer_email}">${bookingData.customer_email}</a></div>

    <h3>Trip Details</h3>
    <div class="field"><span class="label">Pickup:</span> ${bookingData.pickup_location}</div>
    <div class="field"><span class="label">Dropoff:</span> ${bookingData.dropoff_location}</div>
    <div class="field"><span class="label">Date:</span> ${bookingData.trip_date}</div>
    <div class="field"><span class="label">Time:</span> ${bookingData.trip_time}</div>
    <div class="field"><span class="label">Passengers:</span> ${bookingData.passenger_count}</div>
    ${bookingData.trip_type ? `<div class="field"><span class="label">Trip Type:</span> ${bookingData.trip_type}</div>` : ''}
    ${bookingData.special_requests ? `<div class="field"><span class="label">Special Requests:</span> ${bookingData.special_requests}</div>` : ''}

    <h3>Quick Actions</h3>
    <p>
      <a href="tel:${bookingData.customer_phone}" class="button">📞 Call Customer</a>
      <a href="sms:${bookingData.customer_phone}" class="button" style="background: #2196F3;">💬 Text Customer</a>
      <a href="https://customer.moovs.app/lake-ride-pros/new/" class="button" style="background: #FF9800;">🚗 Create in Moovs</a>
    </p>

    <hr style="margin: 30px 0; border: none; border-top: 1px solid #ddd;">
    <p style="color: #666; font-size: 12px;">
      Booking ID: ${bookingId}<br>
      Submitted: ${new Date().toLocaleString()}<br>
      Source: AI Chatbot Widget
    </p>
  </div>
</body>
</html>
`;
```

### Task 3: Create SMS Handler for Customer Confirmation

Location: Create new file `functions/sendBookingConfirmationSMS.js`

What to do:
1. Use existing Twilio/SMS infrastructure pattern from `smsOnCreateV2`
2. Send confirmation to customer's phone
3. Keep message under 160 characters for single SMS

SMS message template:
```
Hi ${firstName}!

Lake Ride Pros here 🚗

We got your booking request:
📍 ${shortPickup} → ${shortDropoff}
📅 ${date} at ${time}

We'll text you pricing & confirmation within 15 min.

Questions? Reply to this text!

- Lake Ride Pros Team
```

### Task 4: Update chatbotQuery Handler Logic

Location: `functions/chatbotQuery.js` - update the main handler

What to do:
1. Pass BOOKING_TOOL to queryOpenAI when appropriate
2. Check response for tool_calls
3. Validate booking data
4. Generate unique booking ID
5. Store in Firestore
6. Send email to owners
7. Send SMS to customer
8. Return success response with booking ID

Handler logic:
```javascript
// Query OpenAI with function calling enabled
const tools = [BOOKING_TOOL];
const data = await queryOpenAI(aiSettings, messages, tools);

const response = data.choices?.[0]?.message;

// Check if GPT wants to submit booking
if (response.tool_calls && response.tool_calls.length > 0) {
  const toolCall = response.tool_calls[0];

  if (toolCall.function.name === "submit_booking_request") {
    const bookingData = JSON.parse(toolCall.function.arguments);

    // Validate required fields
    const validation = validateBookingData(bookingData);
    if (!validation.valid) {
      return res.status(200).json({
        success: true,
        reply: `I need a bit more info: ${validation.missing.join(', ')}. Can you provide that?`,
        bookingSubmitted: false
      });
    }

    // Generate unique booking ID
    const bookingId = `LRP-${Date.now()}-${Math.random().toString(36).substr(2, 9).toUpperCase()}`;

    // Store in Firestore
    await db.collection('bookingRequests').doc(bookingId).set({
      ...bookingData,
      bookingId,
      status: 'pending',
      source: 'chatbot',
      createdAt: admin.firestore.FieldValue.serverTimestamp(),
      updatedAt: admin.firestore.FieldValue.serverTimestamp()
    });
    // Send email to owners (using your existing Gmail function)
    await sendBookingRequestEmail(bookingId, bookingData);

    // Send SMS to customer (using your existing SMS function)
    await sendBookingConfirmationSMS(bookingData);

    // Return success with formatted message
    return res.status(200).json({
      success: true,
      reply: `Perfect! I've sent your request to our team, ${bookingData.customer_name}.

Here's your booking summary:
📍 Pickup: ${bookingData.pickup_location}
📍 Dropoff: ${bookingData.dropoff_location}
📅 Date: ${bookingData.trip_date}
⏰ Time: ${bookingData.trip_time}
👥 Passengers: ${bookingData.passenger_count}
${bookingData.special_requests ? `📝 Notes: ${bookingData.special_requests}\n` : ''}

You'll receive a text message within 15 minutes with pricing and confirmation.

Your booking reference: ${bookingId}`,
      bookingSubmitted: true,
      bookingId: bookingId,
      botName: chatbotSettings.name
    });
  }
}

// Regular conversation - return GPT response
return res.status(200).json({
  success: true,
  reply: response.content,
  bookingSubmitted: false,
  botName: chatbotSettings.name
});
```

### Task 5: Add Response Validation (Guardrails)

Location: `functions/chatbotQuery.js`

What to do:
Add validation function to catch GPT-4o-mini hallucinations about pricing, availability, or services.

```javascript
function validateResponse(userMessage, botResponse) {
  const dangerPatterns = [
    // Pricing hallucinations
    { pattern: /\$\d+/i, reason: "mentioned_price", fallback: "Let me connect you with our team for exact pricing." },
    { pattern: /cost.*\d+.*dollars?/i, reason: "mentioned_price" },
    { pattern: /price.*\d+/i, reason: "mentioned_price" },

    // Availability claims
    { pattern: /we (have|don't have) .* available/i, reason: "claimed_availability", fallback: "Let me have our team check availability for you." },
    { pattern: /(yes|no),? we can (do|provide)/i, reason: "definitive_commitment" },

    // Invented policies
    { pattern: /our policy (is|states)/i, reason: "policy_claim", fallback: "Let me connect you with our team for policy details." },
    { pattern: /guaranteed|promise|definitely can/i, reason: "overconfident", fallback: "Let me have our team confirm that for you." }
  ];

  for (const {pattern, reason, fallback} of dangerPatterns) {
    if (pattern.test(botResponse)) {
      logger.warn("Response validation failed", {
        reason,
        userMessage,
        botResponse: botResponse.substring(0, 100)
      });
      return {
        safe: false,
        reason,
        fallback: fallback || "I want to make sure you get accurate information. Let me connect you with our team on Messenger."
      };
    }
  }

  return { safe: true };
}

// Use in handler BEFORE returning response
const validation = validateResponse(message, response.content);
if (!validation.safe) {
  // Log dangerous response for review
  logger.warn("Blocked potentially inaccurate response", {
    validation_reason: validation.reason,
    original_response: response.content
  });

  return res.status(200).json({
    success: true,
    reply: validation.fallback,
    shouldEscalate: true,
    escalationReason: validation.reason
  });
}
```

### Task 6: Update Widget to Show Booking Success

Location: `public/chatbot-embed.js`

What to do:
1. Detect when `bookingSubmitted: true` in response
2. Show success animation/banner
3. Display booking reference number
4. Offer post-booking options

Add to widget:
```javascript
// In sendMessage function, after receiving response
if (data.bookingSubmitted) {
  appendMessage('assistant', data.reply);
  showBookingSuccess(data.bookingId);
  showPostBookingOptions();
} else {
  appendMessage('assistant', data.reply);
}

function showBookingSuccess(bookingId) {
  const successDiv = document.createElement('div');
  successDiv.className = 'lrp-booking-success';
  successDiv.innerHTML = `
    <div class="lrp-success-icon">✅</div>
    <div class="lrp-success-text">
      <strong>Booking Request Sent!</strong>
      <p>We'll text you within 15 minutes</p>
      <p class="lrp-booking-ref">Ref: ${bookingId}</p>
    </div>
  `;
  messagesContainer.appendChild(successDiv);
}

function showPostBookingOptions() {
  const optionsDiv = document.createElement('div');
  optionsDiv.className = 'lrp-post-booking-options';
  optionsDiv.innerHTML = `
    <p>Need anything else?</p>
    <button onclick="location.reload()">📅 Book Another Ride</button>
```

    <button onclick="window.open('https://m.me/lakeridepros')">💬 Chat on Messenger</button>
  `;
  messagesContainer.appendChild(optionsDiv);
}

Add CSS:
```css
.lrp-booking-success {
  display: flex;
  align-items: center;
  gap: 12px;
  background: #d4edda;
  border: 1px solid #c3e6cb;
  border-radius: 8px;
  padding: 16px;
  margin: 12px 0;
}

.lrp-success-icon {
  font-size: 32px;
}

.lrp-success-text strong {
  color: #155724;
  font-size: 16px;
}

.lrp-booking-ref {
  font-size: 11px;
  opacity: 0.7;
  font-family: monospace;
}

.lrp-post-booking-options {
  text-align: center;
  padding: 16px;
  border-top: 1px solid #eee;
  margin-top: 12px;
}

.lrp-post-booking-options button {
  margin: 5px;
  padding: 10px 20px;
  border: none;
  border-radius: 6px;
  cursor: pointer;
  background: #4CAF50;
  color: white;
  font-size: 14px;
}
```

### Task 7: Add Firestore Collection for Tracking

Location: Firestore Database

What to do:
Create new collection structure for `bookingRequests`:

```javascript
// Collection: bookingRequests
// Document ID: bookingId (e.g., "LRP-1730342567890-X7K9P2")
{
  bookingId: "LRP-1730342567890-X7K9P2",
  customer_name: "John Smith",
  customer_email: "john@example.com",
  customer_phone: "303-555-1234",
  pickup_location: "1234 Blake St, Denver, CO",
  dropoff_location: "Red Rocks Amphitheatre",
  trip_date: "2024-11-15",
  trip_time: "18:00",
  passenger_count: 2,
  special_requests: "Need car seat for toddler",
  trip_type: "one-way",
  status: "pending", // pending, confirmed, completed, cancelled
  source: "chatbot",
  createdAt: Timestamp,
  updatedAt: Timestamp,

  // Optional: Add later for tracking
  responseTime: null, // How long until owner responded
  conversionStatus: null // Did it turn into actual booking?
}
```

### Task 8: Update functions/index.js

Location: `functions/index.js`

What to do:
Add new function exports:

```javascript
// Add these lines with your other attach() calls
attach("sendBookingRequestEmail", "./sendBookingRequestEmail", "sendBookingRequestEmail");
attach("sendBookingConfirmationSMS", "./sendBookingConfirmationSMS", "sendBookingConfirmationSMS");
```

## Testing Checklist

Before deploying to production:

### Unit Tests
- [ ] Booking tool schema validates correctly
- [ ] Response validation catches pricing mentions
- [ ] Response validation catches availability claims
- [ ] Email formatting looks good in Gmail/Outlook
- [ ] SMS is under 160 characters and readable

### Integration Tests
- [ ] Happy path: User provides all info → booking submitted
- [ ] Missing fields: Bot asks for missing info naturally
- [ ] Validation: Invalid email format → bot asks again
- [ ] Guardrails: Bot mentions price → response blocked
- [ ] Email delivery: Check owners@lakeridepros.com inbox
- [ ] SMS delivery: Check customer phone receives message
- [ ] Firestore: Document created with correct structure
- [ ] Widget: Success banner shows with booking ID

### Edge Cases
- [ ] User says "tomorrow" → bot calculates correct date
- [ ] User says "7pm" → bot converts to 19:00
- [ ] User tries booking in past → bot catches it
- [ ] Special characters in name/location → properly escaped
- [ ] Very long special requests → truncated gracefully
- [ ] OpenAI API timeout → graceful error message
- [ ] Gmail API error → logged and fallback shown
- [ ] SMS API error → logged, email still sent

## Deployment Steps

1. **Install dependencies** (if new packages needed):
   ```bash
   cd functions
   npm install
   ```

2. **Test locally with emulators**:
   ```bash
   ```

   firebase emulators:start

3. **Deploy functions**:
   ```bash
   firebase deploy --only functions:chatbotQuery,functions:sendBookingRequestEmail,functions:sendBookingConfirmationSMS
   ```

4. **Update widget on website** (if changes to public/chatbot-embed.js):
   ```bash
   firebase deploy --only hosting
   ```

5. **Test on production**:
   - Visit your website
   - Open chatbot
   - Complete a test booking
   - Verify email received at owners@lakeridepros.com
   - Verify SMS received on test phone

6. **Monitor for first 24 hours**:
   - Check Firebase Functions logs
   - Review any errors
   - Monitor Firestore bookingRequests collection
   - Track email delivery rate
   - Track SMS delivery rate

## Analytics & Monitoring

Add these logging points:

```javascript
// Log when booking collection starts
logger.info("Booking collection started", { userId: conversationId });

// Log when booking submitted
logger.info("Booking submitted", {
  bookingId,
  conversationLength: conversationHistory.length,
  timeToComplete: Date.now() - conversationStartTime
});

// Log validation failures
logger.warn("Response validation failed", {
  reason,
  userMessage,
  botResponse: botResponse.substring(0, 100)
});

// Log email/SMS delivery
logger.info("Booking notification sent", {
  bookingId,
  emailSent: true,
  smsSent: true
});
```

Weekly metrics to review:
- Total booking requests submitted
- Average conversation length to booking
- Validation failure rate (how often responses blocked)
- Email delivery success rate
- SMS delivery success rate
- Conversion rate (booking requests → actual bookings)
- Most common dropoff locations
- Peak booking request times

## Cost Projections

Per booking request:

- GPT-4o-mini (~8 messages): $0.0012
- Email via Gmail API: $0
- SMS via Twilio: $0.01
- Firestore write: $0 (free tier)
- **Total: ~$0.011 per booking request**

Monthly projections:
- 1,000 conversations → 100 booking requests (10% conversion)
- Cost: ~$1.10/month
- If avg booking value is $100+, ROI is excellent

## Code Quality Standards

Follow existing patterns:
- Use async/await (not callbacks)
- Use logger from firebase-functions/v2
- Handle errors gracefully with try/catch
- Return proper HTTP status codes
- Include descriptive error messages
- Add JSDoc comments for functions
- Follow existing naming conventions

## Security Considerations

- No authentication required (public endpoint) ✓
- CORS enabled for external websites ✓
- Rate limiting via Firebase Functions defaults ✓
- No sensitive data in response payloads ✓
- API keys stored in Firestore (not exposed) ✓
- Email/SMS use server-side credentials ✓

## Support & Troubleshooting

Common issues:

OpenAI API errors:
- Check API key in Firestore appSettings/aiContentGenerator
- Verify billing enabled on OpenAI account
- Check rate limits

Email not received:
- Check Gmail API credentials
- Verify domain-wide delegation set up
- Check spam folder
- Review Firebase Functions logs

SMS not received:
- Check Twilio credentials
- Verify phone number format
- Check SMS balance
- Review Firebase Functions logs

Widget not updating:
- Clear browser cache
- Check CORS headers in Network tab
- Verify API endpoint URLs

## Next Steps After Deployment

1. **Monitor first 10 bookings closely**
   - Are customers providing all needed info?
   - Is email format clear for owners?
   - Is SMS confirmation helpful?

2. **Gather feedback from owners**
   - Is email actionable enough?
   - What info is missing?
   - Response time achievable?

3. **Iterate on prompts**

   - Adjust system prompt based on real conversations
   - Add/remove fields as needed
   - Refine validation rules

4. **Consider future enhancements**
   - Add direct Moovs API integration
   - Implement automated pricing lookup
   - Add availability checking
   - Build admin dashboard for booking requests

## Success Criteria

You'll know this is working when:
- ✅ Users can complete bookings without confusion
- ✅ Owners receive clear, actionable emails
- ✅ Customers get timely SMS confirmations
- ✅ Validation catches >90% of hallucinations
- ✅ Cost stays under $0.02 per conversation
- ✅ Booking submission rate >10% of conversations
- ✅ Zero complaints about inaccurate information

## Ready to Implement?

Review this prompt thoroughly, then execute tasks in order. Test extensively in emulators before deploying to production.

Need clarification on any part? Ask before proceeding.

Good luck! 🚀