Weekly Lottery Feature - Implementation Specification

Feature Overview

Build a weekly lottery system where members can win dining vouchers. The lottery creates engagement, drives visits, and burns nice currency to prevent inflation.

Core Mechanics

Entry Rules:

- Every member gets 1 FREE entry per week (automatic)
- Can purchase additional entries: 200 nice per entry (max 10 purchased per week)
- Visit bonus: +1 entry per restaurant visit (max 3 per week)
- Check-in bonus: +2 entries for checking in via app (once per week)
- Maximum total entries per user per week: 16 (1 base + 10 purchased + 3 visits + 2 check-in)

Prize Structure:

- Weekly Standard (most weeks): $50 dining voucher
- Monthly Special (1st week of month): $100 (1st) + $50 (2nd, 2 winners) + $25 (3rd, 5 winners)
- Quarterly Mega (every 13 weeks): $500 (1st) + $200 (2nd, 2 winners) + $100 (3rd, 5 winners)

Drawing Schedule:

- New lottery starts: Monday 12:00 AM
- Drawing occurs: Sunday 8:00 PM
- Winner announced: Immediately after drawing

Database Schema

Tables to Create

```sql
```

```sql
-- Main lottery drawings table
CREATE TABLE lottery_drawings (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  draw_date TIMESTAMPTZ NOT NULL, -- When the drawing occurs (Sunday 8pm)
  week_start_date DATE NOT NULL, -- Monday of that week
  prize_tier TEXT NOT NULL CHECK (prize_tier IN ('standard', 'monthly', 'quarterly')),
  prize_description TEXT NOT NULL, -- e.g., "$50 Dining Voucher"
  prize_value DECIMAL(10,2) NOT NULL,
  status TEXT NOT NULL CHECK (status IN ('upcoming', 'active', 'drawn', 'awarded')) DEFAULT 'upcoming',
  total_entries INTEGER DEFAULT 0,
  total_participants INTEGER DEFAULT 0,
  winning_ticket_number INTEGER,
  random_seed TEXT, -- For provably fair drawing
  drawn_at TIMESTAMPTZ,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Index for finding current active drawing
CREATE INDEX idx_lottery_drawings_status ON lottery_drawings(status, week_start_date);
CREATE INDEX idx_lottery_drawings_draw_date ON lottery_drawings(draw_date);

-- Lottery entries table
CREATE TABLE lottery_entries (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  drawing_id UUID NOT NULL REFERENCES lottery_drawings(id) ON DELETE CASCADE,
  user_id UUID NOT NULL REFERENCES profiles(id) ON DELETE CASCADE,
  entry_type TEXT NOT NULL CHECK (entry_type IN ('base', 'purchased', 'visit', 'checkin')),
  nice_spent INTEGER, -- NULL for free entries (base, visit, checkin)
  quantity INTEGER NOT NULL DEFAULT 1, -- Number of entries in this transaction
  visit_id UUID REFERENCES visits(id), -- Link to visit if entry_type is 'visit'
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Indexes for fast queries
CREATE INDEX idx_lottery_entries_drawing ON lottery_entries(drawing_id);
CREATE INDEX idx_lottery_entries_user ON lottery_entries(user_id, drawing_id);
CREATE INDEX idx_lottery_entries_type ON lottery_entries(drawing_id, entry_type);

-- Winners table
CREATE TABLE lottery_winners (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  drawing_id UUID NOT NULL REFERENCES lottery_drawings(id),
  user_id UUID NOT NULL REFERENCES profiles(id),
  prize_rank INTEGER NOT NULL DEFAULT 1, -- 1st, 2nd, 3rd place
  prize_description TEXT NOT NULL,
  prize_value DECIMAL(10,2) NOT NULL,
  voucher_code TEXT UNIQUE,
  voucher_expiry_date DATE,
  claimed BOOLEAN DEFAULT FALSE,
  claimed_at TIMESTAMPTZ,
  notified BOOLEAN DEFAULT FALSE,
  notified_at TIMESTAMPTZ,
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_lottery_winners_drawing ON lottery_winners(drawing_id);
CREATE INDEX idx_lottery_winners_user ON lottery_winners(user_id);
CREATE INDEX idx_lottery_winners_voucher ON lottery_winners(voucher_code);

-- Statistics tracking table
CREATE TABLE lottery_stats (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  drawing_id UUID NOT NULL REFERENCES lottery_drawings(id) UNIQUE,
  total_participants INTEGER NOT NULL,
  total_entries INTEGER NOT NULL,
  total_nice_spent INTEGER NOT NULL,
  avg_entries_per_user DECIMAL(5,2),
  entries_purchased INTEGER, -- Count of purchased entries
  entries_visit INTEGER, -- Count of visit bonus entries
  entries_checkin INTEGER, -- Count of check-in bonus entries
  entries_base INTEGER, -- Count of base free entries
  created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_lottery_stats_drawing ON lottery_stats(drawing_id);
```

Existing Tables to Reference

Assume these tables exist in your database:

- `profiles` - User profiles table with columns: id, display_name, email, phone
- `nice_accounts` - Nice balance table with columns: id, user_id, balance
- `points_accounts` - Points balance table
- `visits` - Visit tracking table with columns: id, user_id, location_id, visit_date
- `vouchers` - Voucher redemption table

Backend API Endpoints

1. Get Current Active Drawing

Endpoint: `GET /api/lottery/current`

Response:

```typescript
{
 drawing: {
  id: string
  draw_date: string
  prize_tier: 'standard' | 'monthly' | 'quarterly'
  prize_description: string
  prize_value: number
  total_entries: number
  total_participants: number
  status: 'active'
 }
 user_entries: {
  total: number // Total entries for current user
  breakdown: {
   base: number // Always 1
   purchased: number // 0-10
   visit: number // 0-3
   checkin: number // 0 or 2
  }
  remaining: {
   can_purchase: number // Max 10 - currently purchased
   can_visit: number // Max 3 - current visit bonuses
   can_checkin: boolean // True if haven't checked in this week
  }
 }
 odds: {
  numerator: number // user's total entries
  denominator: number // total pool entries
  percentage: string // "0.32%"
 }
 time_until_draw: string // "2 days, 5 hours"
}
```

Logic:

1. Find active drawing where status='active' AND draw_date > NOW()
2. Count user's entries grouped by entry_type
3. Calculate remaining entry opportunities
4. Calculate odds based on total pool

2. Purchase Lottery Entries

Endpoint: `POST /api/lottery/purchase-entries`

Request Body:

```typescript
{
  quantity: number // 1-10
}
```

Response:

```typescript
{
  success: boolean
  entries_purchased: number
  nice_spent: number
  new_balance: number
  total_entries: number // User's total entries now
  message: string
}
```

Logic:

1. Validate quantity (1-10)
2. Get current active drawing
3. Check user hasn't exceeded purchase limit (10 max)
4. Calculate cost: quantity × 200 nice
5. Check user has sufficient nice balance
6. Begin transaction:
    - Deduct nice from nice_accounts
    - Insert into lottery_entries (entry_type='purchased', nice_spent=cost, quantity)
    - Update lottery_drawings.total_entries
7. Commit transaction
8. Return success response

Error Cases:

- Insufficient nice balance → 400 "Insufficient nice balance"
- Purchase limit exceeded → 400 "Maximum 10 purchased entries per week"

- No active drawing → 404 "No active lottery drawing"
- Invalid quantity → 400 "Quantity must be between 1 and 10"

---

3. Record Visit Bonus Entry

Endpoint: `POST /api/lottery/visit-bonus`

Request Body:

```typescript
{
  visit_id: string // UUID of the visit record
}
```

Response:

```typescript
{
  success: boolean
  entry_awarded: boolean
  total_entries: number
  message: string
}
```

Logic:

1. Get current active drawing
2. Verify visit exists and belongs to user
3. Check visit date is within current lottery week
4. Count existing visit bonus entries for this user/drawing
5. If count < 3:
    - Insert lottery_entry (entry_type='visit', visit_id, quantity=1)
    - Update lottery_drawings.total_entries
    - Send notification: "+1 lottery entry earned!"
6. Else: Return success=true, entry_awarded=false, message="Max visit bonuses reached"

Call this endpoint:

- From POS system after visit is recorded
- From visit tracking API after successful visit creation

---

4. Record Check-in Bonus Entry

Endpoint: `POST /api/lottery/checkin-bonus`

Request Body:

```typescript
{
  location_id: string
}
```

Response:

```typescript
{
  success: boolean
  entries_awarded: number // 0 or 2
  total_entries: number
  message: string
}
```

Logic:

1. Get current active drawing
2. Check if user already has check-in entry for this drawing
3. If no existing check-in entry:
    - Insert lottery_entry (entry_type='checkin', quantity=2)
    - Update lottery_drawings.total_entries by 2
    - Send notification: "+2 lottery entries! Thanks for checking in!"
4. Else: Return entries_awarded=0, message="Already earned check-in bonus this week"

Call this endpoint:

- From check-in feature when user checks in at location via app

---

5. Get User Entry History

Endpoint: `GET /api/lottery/my-entries?drawing_id={id}`

Response:

```typescript
```

```
{
  entries: [
    {
      id: string
      entry_type: 'base' | 'purchased' | 'visit' | 'checkin'
      quantity: number
      nice_spent: number | null
      created_at: string
    }
  ]
  total: number
}
```

---

6. Get Past Winners

Endpoint: `GET /api/lottery/winners?limit=10`

Response:

```typescript
{
  winners: [
    {
      id: string
      user_name: string // Anonymized or full based on privacy settings
      prize_description: string
      prize_value: number
      draw_date: string
      total_entries_in_pool: number
      user_entries: number // How many entries the winner had
    }
  ]
}
```

---

7. Admin: Execute Drawing (Protected - Admin Only)

Endpoint: `POST /api/admin/lottery/execute-drawing`

Request Body:

```typescript
```

```
{
  drawing_id: string
}
```

Response:

```typescript
{
  success: boolean
  winner: {
    user_id: string
    user_name: string
    winning_ticket_number: number
    total_entries: number
  }
  stats: {
    total_participants: number
    total_entries: number
    total_nice_spent: number
  }
}
```

Drawing Algorithm (Provably Fair):

```typescript
```

```
async function executeDrawing(drawingId: string) {
  // 1. Get all entries for this drawing
  const entries = await db
    .from('lottery_entries')
    .select('*')
    .eq('drawing_id', drawingId)

  // 2. Build ticket pool (weighted by quantity)
  // Each entry gets sequential ticket numbers
  const ticketPool: { userId: string; ticketNumber: number }[] = []
  let ticketCounter = 0

  // Group by user first to ensure fair distribution
  const userEntries = new Map<string, number>()
  entries.forEach(entry => {
    const current = userEntries.get(entry.user_id) || 0
    userEntries.set(entry.user_id, current + entry.quantity)
  })

  // Assign ticket numbers
  for (const [userId, count] of userEntries) {
    for (let i = 0; i < count; i++) {
      ticketPool.push({ userId, ticketNumber: ticketCounter++ })
    }
  }

  const totalTickets = ticketCounter

  // 3. Generate cryptographically secure random number
  const randomBytes = crypto.getRandomValues(new Uint32Array(1))
  const randomSeed = randomBytes[0].toString()
  const winningTicket = randomBytes[0] % totalTickets

  // 4. Find winner
  const winner = ticketPool.find(t => t.ticketNumber === winningTicket)

  if (!winner) {
    throw new Error('No winner found - drawing error')
  }

  // 5. Generate voucher code
  const voucherCode = generateVoucherCode() // e.g., "LUCKY50-1234"
  const voucherExpiry = new Date()
  voucherExpiry.setDate(voucherExpiry.getDate() + 30) // 30 days validity

  // 6. Begin transaction
```

```
await db.transaction(async (trx) => {
  // Insert winner record
  await trx.from('lottery_winners').insert({
    drawing_id: drawingId,
    user_id: winner.userId,
    prize_rank: 1,
    prize_description: drawing.prize_description,
    prize_value: drawing.prize_value,
    voucher_code: voucherCode,
    voucher_expiry_date: voucherExpiry
  })

  // Update drawing record
  await trx
    .from('lottery_drawings')
    .update({
      status: 'drawn',
      winning_ticket_number: winningTicket,
      random_seed: randomSeed,
      drawn_at: new Date(),
      total_entries: totalTickets,
      total_participants: userEntries.size
    })
    .eq('id', drawingId)

  // Insert stats
  const purchasedCount = entries
    .filter(e => e.entry_type === 'purchased')
    .reduce((sum, e) => sum + e.quantity, 0)
  const visitCount = entries
    .filter(e => e.entry_type === 'visit')
    .reduce((sum, e) => sum + e.quantity, 0)
  const checkinCount = entries
    .filter(e => e.entry_type === 'checkin')
    .reduce((sum, e) => sum + e.quantity, 0)
  const baseCount = entries
    .filter(e => e.entry_type === 'base')
    .reduce((sum, e) => sum + e.quantity, 0)
  const totalNiceSpent = entries
    .filter(e => e.nice_spent)
    .reduce((sum, e) => sum + (e.nice_spent || 0), 0)

  await trx.from('lottery_stats').insert({
    drawing_id: drawingId,
    total_participants: userEntries.size,
    total_entries: totalTickets,
    total_nice_spent: totalNiceSpent,
      avg_entries_per_user: totalTickets / userEntries.size,
      entries_purchased: purchasedCount,
      entries_visit: visitCount,
      entries_checkin: checkinCount,
      entries_base: baseCount
    })
  })

  // 7. Send notifications (outside transaction)
  await notifyWinner(winner.userId, voucherCode, drawing)
  await notifyNonWinners(drawingId, winner.userId)

  return { winner, totalTickets, winningTicket }
}

function generateVoucherCode(): string {
  const prefix = 'LUCKY'
  const random = Math.floor(Math.random() * 10000).toString().padStart(4, '0')
  return `${prefix}${random}`
}
```

Cron Jobs / Scheduled Tasks

1. Create New Weekly Drawing

Schedule: Every Monday at 12:00 AM

Logic:

```typescript
```

```
async function createWeeklyDrawing() {
  const now = new Date()
  const weekStart = startOfWeek(now, { weekStartsOn: 1 }) // Monday
  const drawDate = addDays(weekStart, 6) // Sunday
  drawDate.setHours(20, 0, 0, 0) // 8:00 PM

  // Determine prize tier
  const weekOfMonth = getWeekOfMonth(now)
  const weekOfQuarter = getWeekOfQuarter(now)

  let prizeTier: 'standard' | 'monthly' | 'quarterly'
  let prizeDescription: string
  let prizeValue: number

  if (weekOfQuarter === 1) {
    prizeTier = 'quarterly'
    prizeDescription = '$500 Dining Voucher'
    prizeValue = 500
  } else if (weekOfMonth === 1) {
    prizeTier = 'monthly'
    prizeDescription = '$100 Dining Voucher'
    prizeValue = 100
  } else {
    prizeTier = 'standard'
    prizeDescription = '$50 Dining Voucher'
    prizeValue = 50
  }

  // Create drawing
  await db.from('lottery_drawings').insert({
    draw_date: drawDate,
    week_start_date: weekStart,
    prize_tier: prizeTier,
    prize_description: prizeDescription,
    prize_value: prizeValue,
    status: 'active'
  })

  // Send notification to all users
  await sendPushNotification({
    title: '🎰 New Weekly Lottery!',
    body: `Win ${prizeDescription}. You have 1 free entry!`,
    data: { screen: 'Lottery' }
  })
}
```

2. Award Base Entries

Schedule: Every Monday at 12:05 AM (after drawing creation)

Logic:

typescript

```typescript
async function awardBaseEntries() {
  // Get this week's active drawing
  const drawing = await db
    .from('lottery_drawings')
    .select('*')
    .eq('status', 'active')
    .single()

  if (!drawing) {
    console.error('No active drawing found')
    return
  }

  // Get all active users (have logged in within last 90 days)
  const activeUsers = await db
    .from('profiles')
    .select('id')
    .gte('last_login_at', subDays(new Date(), 90))

  // Batch insert base entries
  const baseEntries = activeUsers.map(user => ({
    drawing_id: drawing.id,
    user_id: user.id,
    entry_type: 'base',
    quantity: 1
  }))

  await db.from('lottery_entries').insert(baseEntries)

  // Update total entries count
  await db
    .from('lottery_drawings')
    .update({ total_entries: baseEntries.length })
    .eq('id', drawing.id)
}
```

3. Execute Weekly Drawing

Schedule: Every Sunday at 8:00 PM

Logic:

```typescript
async function runWeeklyDrawing() {
  // Find drawing scheduled for now
  const drawing = await db
    .from('lottery_drawings')
    .select('*')
    .eq('status', 'active')
    .lte('draw_date', new Date())
    .single()

  if (!drawing) {
    console.log('No drawing scheduled')
    return
  }

  // Execute drawing
  const result = await executeDrawing(drawing.id)

  console.log(`Drawing complete. Winner: ${result.winner.userId}`)
}
```

Frontend Components

1. Lottery Home Widget (Dashboard)

Component: `components/lottery-home-widget.tsx`

Design:

```typescript
```

```
export function LotteryHomeWidget() {
  const { data, isLoading } = useQuery({
    queryKey: ['lottery', 'current'],
    queryFn: () => fetch('/api/lottery/current').then(r => r.json())
  })

  if (isLoading) return <Skeleton />

  const { drawing, user_entries, time_until_draw } = data

  return (
    <Link href="/lottery">
      <Card className="bg-gradient-to-r from-purple-500 to-pink-500 text-white">
        <CardContent className="p-4">
          <div className="flex justify-between items-start">
            <div>
              <p className="text-sm opacity-90">This Week's Lottery</p>
              <p className="text-2xl font-bold mt-1">{drawing.prize_description}</p>
            </div>
            <div className="text-4xl">🎰</div>
          </div>

          <div className="mt-4 flex justify-between items-end">
            <div>
              <p className="text-xs opacity-75">Your Entries</p>
              <p className="text-3xl font-bold">{user_entries.total}</p>
            </div>
            <div className="bg-white/20 px-3 py-1 rounded-full">
              <p className="text-xs font-medium">⏰ {time_until_draw}</p>
            </div>
          </div>
        </CardContent>
      </Card>
    </Link>
  )
}
```

---

2. Full Lottery Screen

Component: `app/lottery/page.tsx`

Sections:

1. Prize display (large, prominent)
2. User's entry count & odds

3. Ways to get more entries (cards for each method)
4. Purchase entries section with packages (1, 5, 10 entries)
5. Recent winners showcase
6. Rules & FAQ accordion

Key Features:

- Real-time countdown to drawing
- Live entry count (updates when user buys entries)
- Odds calculator
- Purchase flow with confirmation
- Entry history

---

3. Purchase Entries Modal

Component: `components/lottery-purchase-modal.tsx`

Features:

- Package selection (1, 5, 10 entries)
- Show discount for bulk purchases
- Display nice cost and new odds
- Confirmation step
- Success animation with confetti

Validation:

- Check nice balance before allowing purchase
- Check remaining purchase limit
- Show error if insufficient nice or limit reached

---

4. Winner Announcement Screen

Component: `app/lottery/winner/[drawing_id]/page.tsx`

Display:

- Confetti animation
- Winner name (or "You Won!" if current user)
- Winning ticket number

- Prize details
- Voucher code (if winner)
- Stats (total entries, participants)
- "Provably fair" section showing random seed

Notification System

Push Notifications to Send

1. New Lottery Started (Monday 9am)

```typescript
{
  title: '🎰 New Weekly Lottery!',
  body: 'Win a $50 dining voucher. You have 1 free entry!',
  data: { screen: 'Lottery' }
}
```

2. Mid-Week Reminder (Wednesday 3pm)

```typescript
{
  title: '🎲 Lottery Update',
  body: `${totalEntries} entries so far. Boost your odds with nice!`,
  data: { screen: 'Lottery' }
}
```

3. 24-Hour Warning (Saturday 8pm)

```typescript
{
  title: '⏰ 24 Hours Left!',
  body: `Drawing tomorrow night. Your odds: 1 in ${odds}`,
  data: { screen: 'Lottery' }
}
```

4. Final Hour (Sunday 7pm)

```typescript
```

```
{
  title: '🔥 Final Hour!',
  body: 'Drawing at 8pm tonight. Last chance to buy entries!',
  data: { screen: 'Lottery' }
}
```

5. Winner Announcement (Sunday 8:05pm to winner)

```typescript
{
  title: '🎉 CONGRATULATIONS! 🎉',
  body: `You WON! ${prizeDescription}. Code: ${voucherCode}`,
  data: { screen: 'LotteryWinner', drawing_id: drawingId }
}
```

6. Non-Winner (Sunday 8:05pm to non-winners)

```typescript
{
  title: '😳 Not this time...',
  body: 'New lottery starts Monday! Better luck next week 🍀',
  data: { screen: 'Lottery' }
}
```

7. Visit Bonus Earned (at visit time)

```typescript
{
  title: '🎰 Lottery Entry Earned!',
  body: `+1 entry for visiting! You now have ${newTotal} entries.`,
  data: { screen: 'Lottery' }
}
```

Integration Points

1. Visit Tracking System

When a visit is recorded (from POS or manual entry):

```typescript
```

```
// After creating visit record
await fetch('/api/lottery/visit-bonus', {
  method: 'POST',
  body: JSON.stringify({ visit_id: newVisit.id })
})
```

---

2. Check-In System

When user checks in at location:

```typescript
// After successful check-in
await fetch('/api/lottery/checkin-bonus', {
  method: 'POST',
  body: JSON.stringify({ location_id: location.id })
})
```

---

3. Nice Balance System

Purchase entries flow must:

1. Check nice_accounts.balance
2. Deduct nice atomically
3. Record transaction in nice_transactions table

---

4. Voucher System

After drawing, winner gets voucher_code that should:

- Be added to vouchers table
- Have expiry date (30 days from win)
- Be redeemable at POS
- Track usage (claimed boolean)

Edge Cases & Error Handling

1. User Has Insufficient Nice

When: Attempting to purchase entries

Handle:

```typescript
if (userNiceBalance < cost) {
  return {
    error: 'INSUFFICIENT_NICE',
    required: cost,
    current: userNiceBalance,
    shortfall: cost - userNiceBalance,
    message: 'You need ${shortfall} more nice to purchase these entries'
  }
}
```

---

2. Purchase Limit Exceeded

When: User already purchased 10 entries this week

Handle:

```typescript
if (currentPurchased + quantity > 10) {
  return {
    error: 'PURCHASE_LIMIT_EXCEEDED',
    limit: 10,
    current: currentPurchased,
    max_can_buy: 10 - currentPurchased,
    message: 'Maximum 10 purchased entries per week. You can buy ${max_can_buy} more.'
  }
}
```

---

3. Visit Bonus Limit Exceeded

When: User already has 3 visit bonuses this week

Handle:

- Still record the visit (don't block visit tracking)
- Just don't award lottery entry

- Show message: "Maximum visit bonuses reached for this week"

---

4. No Active Drawing

When: No drawing with status='active'

Handle:

- Show message: "No active lottery at this time. Check back Monday!"
- Disable purchase buttons
- Show countdown to next lottery

---

5. Drawing Has No Entries

When: Drawing time arrives but total_entries = 0

Handle:

- Don't execute drawing
- Roll over prize to next week (increase prize value)
- Log error for investigation

---

6. Multiple Winners with Same Ticket (Collision)

When: Ticket pool has duplicates (shouldn't happen with correct logic)

Handle:

- This shouldn't be possible with sequential ticket numbering
- But add safety check: if duplicate found, re-draw with new random number
- Log incident for investigation

Testing Requirements

Unit Tests

1. Drawing Algorithm:

- Test with 10 entries, verify each user has proportional chance
- Test with 1,000 entries, verify distribution is fair

- Test with single entry (should always win)
- Test with 0 entries (should throw error)

2. Entry Purchase:

- Test successful purchase (deducts nice, creates entry)
- Test insufficient balance (returns error, no deduction)
- Test limit exceeded (returns error)
- Test concurrent purchases (no race condition)

3. Odds Calculation:

- Test with various entry counts
- Test edge case: user has all entries (100% odds)
- Test edge case: user has 1 of 10,000 entries

Integration Tests

1. Full Lottery Cycle:

- Create drawing → Award base entries → User buys entries → Execute drawing → Verify winner

2. Visit Bonus Flow:

- Record visit → Check entry awarded → Verify limit enforcement

3. Multi-Week Scenario:

- Week 1 drawing → Week 2 new drawing → Verify entries don't carry over

E2E Tests

1. User Journey:

- Login → View lottery → Buy entries → Visit restaurant (bonus) → Check entries updated → Wait for drawing → See result

---

Admin Dashboard Features

Pages to Create:

1. Current Lottery Overview

- Prize details
- Entry counts by type

- Revenue (nice spent)
- Top participants
- Execute drawing button (if draw time passed)

2. Lottery History

- Past drawings table
- Winner details
- Statistics per drawing
- Total nice burned (all time)

3. Create Manual Drawing

- Form to create special lottery (holidays, events)
- Set custom prize
- Set draw date
- Override prize tier

Performance Considerations

1. Entry Counting

Problem: With 10,000 users, counting entries could be slow

Solution:

- Maintain `total_entries` count on lottery_drawings table
- Increment on each entry insert (not COUNT query)
- Use database triggers or application-level atomic increment

2. Drawing Execution

Problem: With 100,000+ entries, building ticket pool could be memory-intensive

Solution:

- Process in batches of 1,000 entries
- Use streaming approach for large pools
- Consider alternative: Store cumulative ticket ranges per user

3. Notifications

Problem: Sending 10,000 push notifications could be slow

Solution:

- Use batch sending (chunks of 500)
- Queue notifications (don't block API response)
- Send winner notification first (immediate)
- Send non-winner notifications in background

Analytics to Track

Metrics Dashboard:

- Participation rate (% of active users who have entries)
- Average entries per participant
- Nice consumption per week (entries purchased × 200)
- Visit attribution (visits driven by lottery)
- Check-in rate increase (before/after lottery)
- ROI: (incremental revenue - prize cost) / prize cost

User Segmentation:

- Power participants (10+ entries every week)
- Casual participants (1-2 entries)
- Non-participants (never bought entries)

Security Considerations

1. Prevent Entry Manipulation

- Use database constraints (max 10 purchased, max 3 visit bonuses)
- Validate on both frontend AND backend
- Use transactions to prevent race conditions

2. Prevent Nice Exploitation

- Verify nice balance before deducting
- Use atomic transactions (balance check + deduction in single transaction)
- Log all nice transactions for audit trail

3. Drawing Fairness

- Use crypto.getRandomValues() (not Math.random())
- Store random seed for provability
- Log all drawing execution details
- Allow public verification of results

4. Admin Access

- Protect /api/admin/* endpoints with role check
- Log all admin actions (who executed drawing, when)
- Require 2FA for executing drawings

Rollout Strategy

Phase 1: Soft Launch (Week 1)

- Enable for 10% of users
- Prize: $25 (lower risk)
- Monitor for bugs
- Gather feedback

Phase 2: Expansion (Week 2-3)

- Enable for 50% of users
- Prize: $50
- Fix any issues found
- Optimize performance

Phase 3: Full Launch (Week 4+)

- Enable for all users
- Full prize schedule (standard/monthly/quarterly)
- Marketing push
- Track metrics

Success Criteria

Metrics to Hit:

- 30%+ participation rate (3,000+ participants if 10,000 users)
- 5,000+ entries per week
- 500,000+ nice burned per week
- 300+ extra visits attributed to lottery
- 5× increase in app opens on lottery draw days

Code Quality:

- 80%+ test coverage on lottery logic
- Zero race conditions in entry creation
- Sub-200ms API response times
- Provably fair drawings (verifiable randomness)

Additional Notes

Tech Stack Assumptions:

- Next.js 14 (App Router)
- Supabase (PostgreSQL + Auth)
- React Query for data fetching
- Tailwind CSS for styling
- TypeScript throughout

Existing System Context:

- Nice currency exists (nice_accounts table)
- Points currency exists (points_accounts table)
- Visit tracking exists (visits table)
- Push notifications system exists
- User profiles exist (profiles table)

Nice Currency:

- 4 nice = 1 point conversion exists
- Users generate nice passively (2 nice/hour base rate)
- Visit multipliers apply (5×, 10×, 15× based on frequency)

Implementation Order

Suggested sequence:

1. **Database setup** (Day 1)
   - Create all tables
   - Add indexes
   - Test with sample data
2. **Core API endpoints** (Day 2-3)
   - GET /api/lottery/current
   - POST /api/lottery/purchase-entries
   - POST /api/lottery/visit-bonus
   - POST /api/lottery/checkin-bonus
3. **Drawing logic** (Day 4)
   - Implement executeDrawing function
   - Test with various entry counts
   - Verify fairness and randomness
4. **Cron jobs** (Day 5)
   - Create weekly drawing (Monday)
   - Award base entries (Monday)
   - Execute drawing (Sunday)
5. **Frontend components** (Day 6-8)
   - Home widget
   - Full lottery screen
   - Purchase flow
   - Winner announcement
6. **Notifications** (Day 9)
   - All push notification triggers
   - Email notifications (optional)
7. **Testing** (Day 10)
   - Unit tests
   - Integration tests
   - E2E tests
8. **Admin dashboard** (Day 11-12)
   - Lottery overview

- Execute drawing UI
- History and stats

9. **Polish & optimization** (Day 13-14)
    - Performance tuning
    - Error handling
    - UI/UX improvements

Questions to Clarify

Before implementing, please confirm:

1. **Existing cron infrastructure:** Do you have a job scheduler (Vercel Cron, node-cron, etc.)?
2. **Push notification system:** What service are you using (Firebase, OneSignal, Expo)?
3. **Visit tracking:** How are visits currently recorded (POS integration, manual, app check-in)?
4. **Voucher system:** Do you have an existing voucher/coupon table structure?
5. **User privacy:** Should winner names be anonymized publicly? (e.g., "Sarah T." instead of full name)
6. **Admin roles:** What role should be able to execute drawings? (super_admin, manager, etc.)

Summary

This lottery feature will:

- ✅ Drive weekly engagement (countdown creates urgency)
- ✅ Burn nice currency (prevents inflation)
- ✅ Increase visits (visit bonus incentive)
- ✅ Increase app usage (check-in bonus)
- ✅ Cost minimal ($8/week real cost for $50 voucher)
- ✅ Provide social proof (winner announcements)
- ✅ Create FOMO (limited time, public odds)

Estimated development time: 2 weeks for full implementation with testing

Expected ROI: 500+ incremental visits/week, 5× app engagement increase, cost of $8/week

Now build it! 🚀