SPÉCIFICATIONS COMPLÈTES - PAGE COMMANDER AMBIANCE+

TABLE DES MATIÈRES

1. Vue d'Ensemble
2. Architecture Interface
3. Système Menu Dynamique
4. Logique Panier & Commande
5. Gestion Stock Temps Réel
6. Workflow Paiement
7. Flux Commande Temps Réel
8. Mode Hors Ligne & Synchronisation
9. Optimisations Performance
10. Métriques & Analytics

VUE D'ENSEMBLE

Objectif Global

Page de commande ultra-rapide pour environnement club nocturne, optimisée pour maximiser le volume de commandes et le CA, avec fonctionnement garanti même en conditions réseau difficiles.

Philosophie Design

- **"Beautiful in Simplicity"** - Zéro élément superflu
- **Terminal Matrix** - Vert néon sur fond noir

- **Mobile-first** - Optimisé pour usage en club
- **Performance critique** - < 20KB total, < 2s chargement

Contraintes Techniques

- Réseau saturé en club (WiFi/4G instable)
- Utilisation dans l'obscurité
- Écrans tactiles avec gants/mains mouillées
- Musique forte (pas de dépendance audio)

ARCHITECTURE INTERFACE

Spécifications Visuelles

- **Couleurs** : Vert néon (`#00FF41`) sur fond noir (`#000000`)
- **Police** : Monospace (Courier New, Monaco, SF Mono)
- **Aucun emoji/icône** : Texte pur uniquement
- **Bordures** : 2px solid `#00FF41` pour effet terminal
- **Animations** : Hover subtils, pas de surcharge

Structure Interface

```
|                                          |
| BIERES                                   |
|   HEINEKEN          6.50€    [AJOUTER]   |
|   CORONA            7.00€    [AJOUTER]   |
|   LEFFE BLONDE - RUPTURE STOCK  [INDISPO] |
|                                          |
| SHOTS                                    |
| BOUTEILLES VIP                           |
|                                          |
├──────────────────────────────────────────┤
| PANIER (2 articles)              31.50€ |
| 2x MOJITO ROYAL                  25.00€ |
| 1x HEINEKEN                       6.50€ |
|          [APPLE PAY - 31.50€]            |
└──────────────────────────────────────────┘
```

Responsive Design

- **Desktop** : 400px max-width centré
- **Mobile** : Pleine largeur avec bordures supprimées
- **Tactile** : Boutons minimum 44px (Apple guidelines)
- **Zone de scroll** : Optimisée pour pouce

SYSTÈME MENU DYNAMIQUE

Structure Catégories

javascript

```javascript
const menuStructure = {
  categories: [
    { name: 'COCKTAILS', priority: 1, collapsible: true },
    { name: 'BIERES', priority: 2, collapsible: true },
    { name: 'SHOTS', priority: 3, collapsible: true },
    { name: 'BOUTEILLES VIP', priority: 4, collapsible: true },
    { name: 'SOFTS', priority: 5, collapsible: true }
  ]
};
```

Gestion Multi-Club

Chaque club peut personnaliser complètement son menu :

javascript

```javascript
const clubMenuStructure = {
  club_id: "le-pacha-marseille",
  menu_items: [
    {
      id: "cocktail-001",
      custom_name: "MOJITO ROYAL",       // Nom personnalisé par le club
      base_name: "mojito",               // Référence système
      description: "Rhum premium, menthe fraiche", // Modifiable
      price: 12.50,
      category: "cocktails",
      available: true,
      preparation_time: 300,             // secondes
      allergens: ["none"],
      alcohol_content: 12.5
    }
  ]
};
```

Format Item Menu Standard

- **ID unique** : Identifiant système permanent
- **Nom personnalisé** : Modifiable par le club via dashboard
- **Description** : Personnalisable (ingrédients, style)
- **Prix** : Ajustable par club
- **Disponibilité** : Gestion stock temps réel
- **Catégorie** : Organisationnelle

API Modification Menu

javascript

```
// Endpoint pour modification par le gérant
PUT /api/clubs/{clubId}/menu/{itemId}
{
  "custom_name": "MOJITO PREMIUM",
  "description": "Rhum XO, menthe bio, citron vert",
  "price": 15.00
}
```

LOGIQUE PANIER & COMMANDE

Classe CartManager

javascript

```
class CartManager {
  constructor() {
    this.cart = JSON.parse(localStorage.getItem('cart') || '[]');
    this.maxQuantityPerItem = 10; // Sécurité anti-erreur uniquement
  }

  addItem(itemId, itemData) {
    const existingItem = this.cart.find(item => item.id === itemId);

    if (existingItem) {
      if (existingItem.quantity < this.maxQuantityPerItem) {
        existingItem.quantity++;
      } else {
        throw new Error(`Maximum ${this.maxQuantityPerItem} par article`);
      }
    } else {
      this.cart.push({
        id: itemId,
        name: itemData.name,
        price: itemData.price,
        quantity: 1,
        addedAt: Date.now()
      });
    }

    this.saveCart();
    this.updateUI();
  }
}
```

Stratégie Maximisation Volume

- **Aucune limite** sur nombre total de commandes
- **Aucune limite** sur montant total

- **Aucun minimum** de commande
- **Seule limite** : 10 par article (protection anti-erreur)
- **Encouragement** : Suggestions cross-sell, recommande rapide

Interface Quantités

- **Contrôles tactiles** : Boutons [-] [quantité] [+] de 35px minimum
- **Passage automatique** : AJOUTER → contrôles quantité dès première sélection
- **Calcul temps réel** : Prix total mis à jour instantanément
- **Persistance** : Sauvegarde localStorage à chaque modification

---

GESTION STOCK TEMPS RÉEL

WebSocket Stock Updates

```javascript
const stockSocket = new WebSocket(`wss://api.ambiance-plus.fr/stock/${clubId}`);

stockSocket.onmessage = (event) => {
  const stockUpdate = JSON.parse(event.data);

  stockUpdate.items.forEach(item => {
    updateItemAvailability(item.id, item.available, item.stock_level);
  });
};
```

États de Disponibilité

- **Disponible** : Bouton AJOUTER actif, couleur normale
- **Stock bas** : Mention discrète, pas de blocage
- **Rupture stock** : Bouton INDISPO grisé, article non sélectionnable

- **Retour stock** : Réactivation automatique

Validation Stock Commande

javascript

```javascript
async function validateCartStock(cart) {
  const stockCheck = await fetch('/api/stock/validate', {
    method: 'POST',
    body: JSON.stringify({ items: cart })
  });

  const result = await stockCheck.json();

  if (!result.valid) {
    showStockConflict(result.unavailable_items);
    return false;
  }

  return true;
}
```

Gestion Conflits Stock

- **Détection** : Validation avant paiement
- **Résolution** : Suppression auto articles indisponibles + notification
- **Alternative** : Proposition produits similaires si disponible

---

WORKFLOW PAIEMENT

Système Multi-Paiement avec Fallback

javascript

```
class PaymentManager {
  constructor() {
    this.paymentMethods = this.detectAvailableMethods();
    this.fallbackChain = ['apple_pay', 'google_pay', 'card', 'sms'];
  }

  async processWithFallback(total, cart) {
    for (const method of this.fallbackChain) {
      try {
        if (this.paymentMethods.includes(method)) {
          const result = await this.processSingleMethod(method, total, cart);
          if (result.success) return result;
        }
      } catch (error) {
        console.log(`Payment method ${method} failed, trying next...`);
        continue;
      }
    }

    throw new Error('All payment methods failed');
  }
}
```

Méthodes de Paiement

1. **Apple Pay** (priorité) : Touch ID/Face ID, intégration native
2. **Google Pay** : Authentification biométrique Android
3. **Carte bancaire** : Stripe Elements, 3D Secure
4. **SMS Fallback** : En cas d'échec total, commande par SMS

Optimisations Performance Paiement

- **Préchargement** : Stripe initialisé dès scan QR

- **Cache** : Éléments de paiement en mémoire
- **Timeout** : 30 secondes max par méthode
- **Retry automatique** : 3 tentatives par méthode

Sécurité

- **Validation côté serveur** : Double vérification montant
- **Rate limiting** : 5 tentatives max par 10 minutes
- **Logs complets** : Traçabilité échecs paiement
- **PCI Compliance** : Aucune donnée carte stockée

FLUX COMMANDE TEMPS RÉEL

États de Commande

1. **payment_processing** : Traitement paiement en cours
2. **payment_confirmed** : Paiement réussi → Affichage "EN COURS DE PRÉPARATION"
3. **order_ready** : Barman termine → Affichage "COMMANDE PRÊTE"
4. **order_completed** : Client récupère (optionnel)

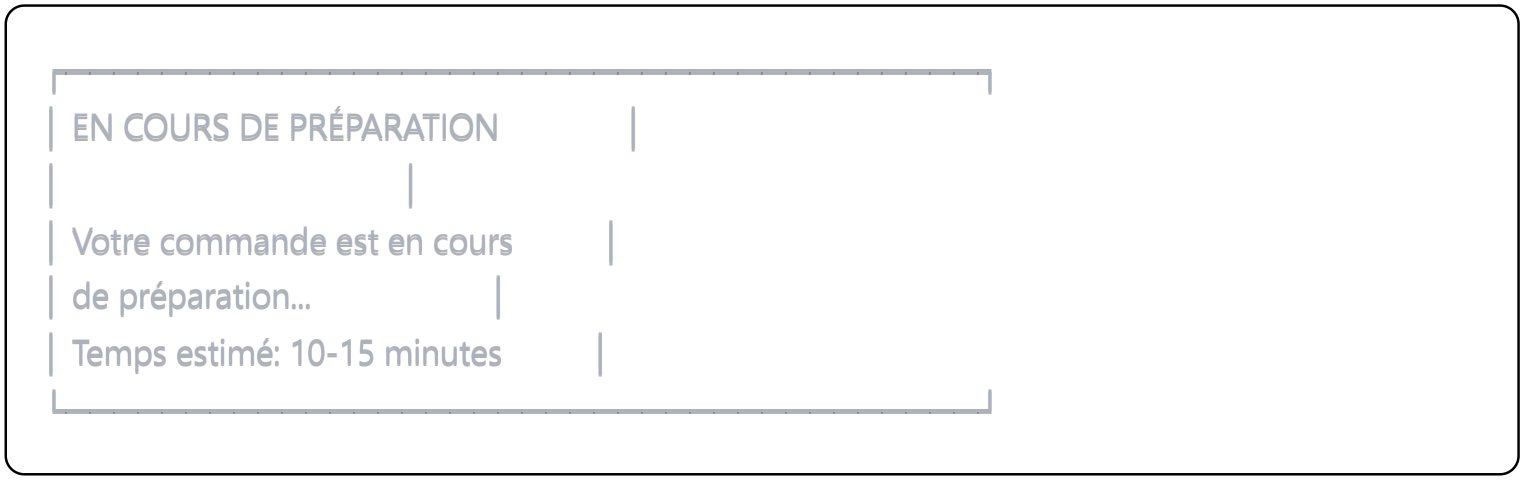
Côté Client - Suivi Temps Réel

```javascript
```

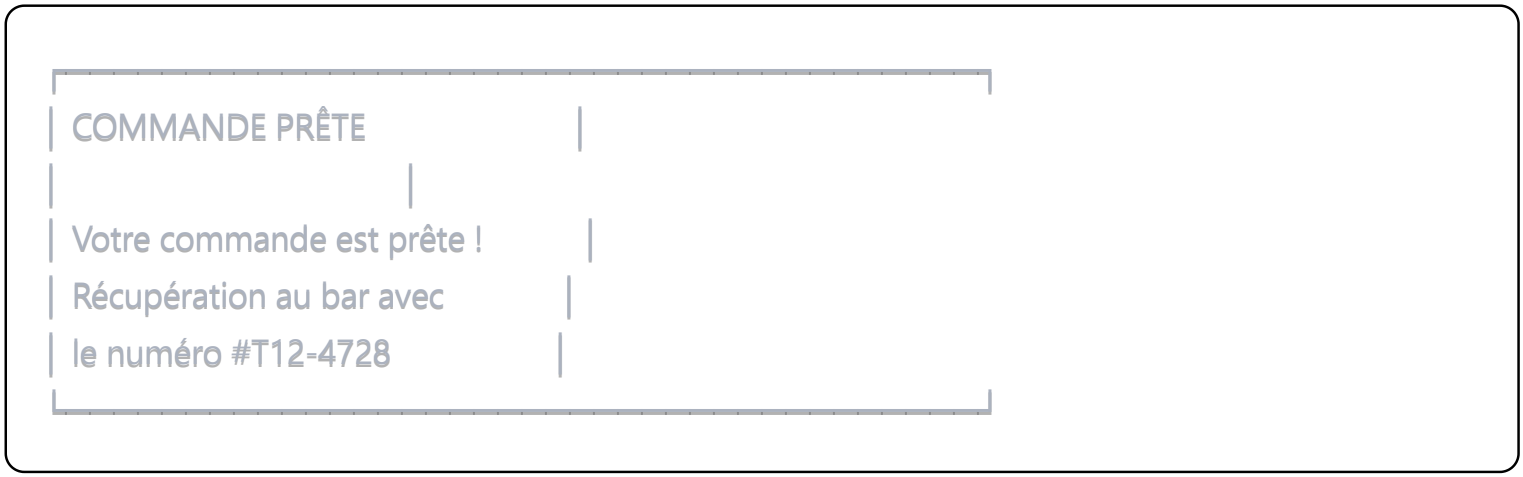
```
class OrderStatusManager {
  constructor(orderNumber) {
    this.orderNumber = orderNumber;
    this.currentStatus = 'payment_processing';
    this.initWebSocket();
  }

  updateOrderStatus(newStatus, data) {
    switch(newStatus) {
      case 'payment_confirmed':
        this.showPreparationStatus();
        break;

      case 'order_ready':
        this.showReadyStatus(data.pickup_time);
        break;
    }

    this.currentStatus = newStatus;
  }

  showPreparationStatus() {
    document.getElementById('statusPreparing').style.display = 'block';
    document.getElementById('statusReady').style.display = 'none';
    // Animation pulse
  }

  showReadyStatus(pickupTime) {
    document.getElementById('statusPreparing').style.display = 'none';
    document.getElementById('statusReady').style.display = 'block';
    // Animation glow + vibration
  }
}
```

Interface Client - États Visuels

État Préparation :



État Prêt :



Côté Barman - Interface de Gestion

javascript

```
class BarmanInterface {
  addNewOrder(order) {
    const orderCard = `
      <div class="order-card" data-order-id="${order.id}">
        <div class="order-header">
          <span class="order-number">${order.number}</span>
          <span class="order-table">${order.table}</span>
        </div>

        <div class="order-items">
          ${order.items.map(item => `${item.quantity}x ${item.name}`).join('<br>')}
        </div>

        <div class="order-actions">
          <button onclick="this.startPreparation('${order.id}')" class="btn-start">
            COMMENCER
          </button>
          <button onclick="this.markReady('${order.id}')" class="btn-ready" disabled>
            TERMINÉ
          </button>
        </div>
      </div>
    `;
  }

  markReady(orderId) {
    // Notification automatique au client
    this.socket.send(JSON.stringify({
      type: 'order_ready',
      order_id: orderId,
      ready_at: Date.now()
    }));
  }
}
```

Architecture WebSocket Serveur

javascript

```
class OrderOrchestrator {
  // Paiement confirmé
  async onPaymentConfirmed(orderData) {
    const order = await this.createOrder(orderData);

    // 1. Confirmer au client
    clientSocket.emit('order_status', {
      status: 'payment_confirmed',
      order_number: order.number
    });

    // 2. Notifier le staff
    staffSockets.forEach(socket => {
      socket.emit('new_order', { order });
    });
  }

  // Barman marque prêt
  async onOrderReady(orderId) {
    // Notification immédiate au client
    clientSocket.emit('order_status', {
      status: 'order_ready',
      order_number: order.number,
      pickup_location: 'BAR'
    });
  }
}
```

MODE HORS LIGNE & SYNCHRONISATION

Stratégie Cache Agressif

javascript

```javascript
// Service Worker pour fonctionnement offline
self.addEventListener('fetch', (event) => {
  if (event.request.url.includes('/api/menu/')) {
    event.respondWith(
      caches.open('ambiance-menu-v1').then(cache => {
        return cache.match(event.request).then(cachedResponse => {
          if (cachedResponse) {
            // Retourner cache immédiatement
            return cachedResponse;
          }

          // Premier chargement
          return fetch(event.request).then(response => {
            cache.put(event.request, response.clone());
            return response;
          });
        });
      })
    );
  }
});
```

Gestion Actions Hors Ligne

javascript

```
class OfflineManager {
  addToCartOffline(item) {
    // Ajout local immédiat
    this.cart.push(item);
    this.updateLocalUI();

    // Queue pour synchronisation ultérieure
    this.pendingActions.push({
      type: 'ADD_TO_CART',
      data: item,
      timestamp: Date.now(),
      synced: false
    });

    this.savePendingActions();
  }

  async syncPendingActions() {
    const unsynced = this.pendingActions.filter(action => !action.synced);

    for (const action of unsynced) {
      try {
        await this.syncAction(action);
        action.synced = true;
      } catch (error) {
        break; // Arrêter si échec réseau
      }
    }
  }
}
```

Fonctionnalités Offline

- **Consultation menu** : 100% disponible (cache local)

- **Ajout panier** : Immédiat, sync différée
- **Modification quantités** : Local, sync au retour réseau
- **Consultation commandes** : Affichage dernier état connu
- **Paiement** : Nécessite réseau (sécurité)

OPTIMISATIONS PERFORMANCE

Target Bundle Ultra-Léger

```
Répartition taille (objectif < 20KB) :
- HTML + CSS inline : 8KB
- JavaScript core : 6KB
- Menu data compressé : 4KB
- Intégration paiement : 2KB
Total : 20KB
```

Compression Menu

javascript

```javascript
function compressMenuData(menu) {
  return menu.map(item => [
    item.id,
    item.name,
    (item.price * 100), // Centimes pour éviter les décimales
    item.available ? 1 : 0,
    item.category_id
  ]);
}
```

Stratégie Chargement

```javascript
async function initApp() {
  // 1. Interface critique (immédiat)
  renderShell();

  // 2. Menu cached (< 100ms)
  const menu = await loadCachedMenu();
  renderMenu(menu);

  // 3. Stock live (background)
  initStockSocket();

  // 4. Payment ready (background)
  preloadPaymentInfrastructure();
}
```

Optimisations Réseau

- **Compression GZIP** : Réduction 70% taille transfert
- **CDN** : Fichiers statiques en edge
- **HTTP/2** : Multiplexing requêtes
- **Preload critical** : Menu et styles en priorité

MÉTRIQUES & ANALYTICS

KPIs Performance Technique

- **Temps scan QR → menu affiché** : < 2 secondes
- **Temps ajout article → panier** : < 500ms

- **Temps commande → paiement** : < 30 secondes
- **Taux de réussite paiement** : > 98%
- **Fonctionnement offline** : 100% consultation, 90% commande
- **Taille bundle** : < 20KB
- **First Contentful Paint** : < 1.5s

KPIs Business

- **Panier moyen** : 35-50€ (sans limite imposée)
- **Commandes par client/soirée** : 3-5 (encouragé)
- **Taux conversion** : > 30% (scan → commande)
- **Volume total/table/soirée** : 150-300€
- **Temps préparation moyen** : 10-15 minutes
- **Taux satisfaction** : > 90%

Tracking Analytics

```javascript
```

```
function trackOrderEvent(eventType, data) {
  analytics.track(eventType, {
    club_id: clubId,
    order_value: data.total,
    items_count: data.items.length,
    payment_method: data.paymentMethod,
    table_number: data.tableNumber,
    timestamp: Date.now(),
    user_agent: navigator.userAgent,
    connection_type: navigator.connection?.effectiveType
  });
}

// Events critiques
trackOrderEvent('menu_loaded', { load_time: performance.now() });
trackOrderEvent('item_added', { item_id, category });
trackOrderEvent('payment_started', { method, amount });
trackOrderEvent('order_confirmed', { order_number, total_time });
```

BASE DE DONNÉES

Tables Principales

sql

```sql
-- Menu personnalisé par club
CREATE TABLE club_menu_items (
  id SERIAL PRIMARY KEY,
  club_id UUID REFERENCES clubs(id),
  base_item_id VARCHAR(50),
  custom_name VARCHAR(100) NOT NULL,
  custom_description TEXT,
  price DECIMAL(8,2) NOT NULL,
  category VARCHAR(50),
  available BOOLEAN DEFAULT true,
  sort_order INTEGER DEFAULT 0,
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Commandes
CREATE TABLE orders (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  order_number VARCHAR(20) UNIQUE NOT NULL,
  club_id UUID REFERENCES clubs(id),
  client_id UUID REFERENCES clients(id),
  table_number INTEGER,
  items JSONB NOT NULL,
  total_amount DECIMAL(8,2) NOT NULL,
  payment_intent_id VARCHAR(100),
  status VARCHAR(20) DEFAULT 'confirmed',
  created_at TIMESTAMP DEFAULT NOW(),
  ready_at TIMESTAMP
);

-- Suivi état des commandes
CREATE TABLE order_status_log (
  id SERIAL PRIMARY KEY,
  order_id UUID REFERENCES orders(id),
  status VARCHAR(50) NOT NULL,
  changed_by VARCHAR(50),
```

```
  changed_at TIMESTAMP DEFAULT NOW(),
  metadata JSONB
);

-- Sessions clients WebSocket
CREATE TABLE client_sessions (
  session_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  order_number VARCHAR(20),
  socket_id VARCHAR(100),
  club_id UUID REFERENCES clubs(id),
  table_number INTEGER,
  created_at TIMESTAMP DEFAULT NOW(),
  last_seen TIMESTAMP DEFAULT NOW()
);
```

SÉCURITÉ

Validations Côté Client

javascript

```
function validateOrder(cart) {
  const errors = [];

  if (cart.length === 0) {
    errors.push('Panier vide');
  }

  cart.forEach(item => {
    if (item.quantity > 10) {
      errors.push(`Maximum 10 ${item.name} par article`);
    }

    if (item.price < 0 || item.price > 1000) {
      errors.push('Prix article invalide');
    }
  });

  return { valid: errors.length === 0, errors };
}
```

Protection Anti-Fraude

- **Rate limiting** : 5 commandes max par 10 minutes
- **Validation serveur** : Double vérification prix et stock
- **Session tracking** : Détection patterns suspects
- **IP filtering** : Blocage temporaire si abus
- **Sanitisation** : Nettoyage données input

Conformité

- **RGPD** : Consentement cookies, droit suppression
- **PCI DSS** : Aucune donnée carte stockée

- **Âge légal** : Présupposé (contrôle entrée club)
- **CGU/CGV** : Acceptation explicite

ÉVOLUTIONS FUTURES

Version 2 (3-6 mois)

- Réservation tables
- Programme fidélité avancé
- Jeux/concours live
- Intégration réseaux sociaux

Version 3 (6-12 mois)

- Multi-établissement
- API publique partenaires
- IA recommandations
- Blockchain/NFT rewards

Ce document constitue la spécification technique complète pour la page Commander d'Ambiance+, garantissant une implémentation robuste et performante adaptée à l'environnement exigeant des clubs nocturnes.