

# CultiReFINE 予約システム画面遷移図

## 1. 全体画面遷移フロー

```

graph TB
    Start[LINEリッチメニュー] --> Auth{LINE ID認証}
    Auth -->|認証成功| MainMenu[メインメニュー画面]
    Auth -->|認証失敗| Error[エラー画面]

    MainMenu --> Reservation[予約フォーム]
    MainMenu --> TicketMgmt[チケット管理]
    MainMenu --> Documents[書籍一覧]

    Reservation --> ToggleProxy{同伴者予約 ?}
    ToggleProxy -->|Yes| SelectVisitor[来院者選択]
    ToggleProxy -->|No| SelectVisitor

    SelectVisitor --> NewVisitor[新規来院者登録]
    SelectVisitor --> RoomType{ペア部屋 ?}

    RoomType -->|Yes| PairRoom[ペア部屋予約画面]
    RoomType -->|No| SingleRoom[通常予約画面]

    PairRoom --> Confirm[予約確認画面]
    SingleRoom --> Confirm

    Confirm --> Complete[予約完了]
    Complete --> ContinueBook{続けて予約 ?}
    ContinueBook -->|Yes| Reservation
    ContinueBook -->|No| MainMenu

    TicketMgmt --> TicketDetail[チケット詳細]
    Documents --> DocView[書類閲覧]

```

## 2. 認証フロー

### 2.1 LINE ID認証プロセス

```

sequenceDiagram
    participant User as ユーザー
    participant LINE as LINEアプリ
    participant GAS as Google Apps Script
    participant Sheet as スプレッドシート

    User->>LINE: リッチメニュータップ
    LINE->>GAS: LINE ID送信
    GAS->>Sheet: LINE ID照合
    Sheet-->>GAS: 法人グループ情報

```

GAS->>GAS: 権限判定 ( 本会員/サブ会員 )

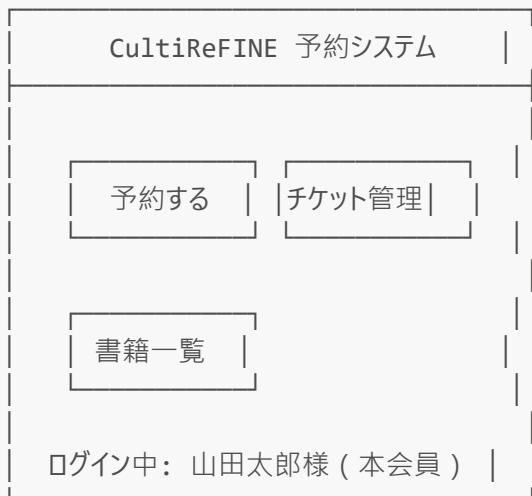
GAS-->>User: 認証済み画面表示

## 2.2 権限判定ロジック

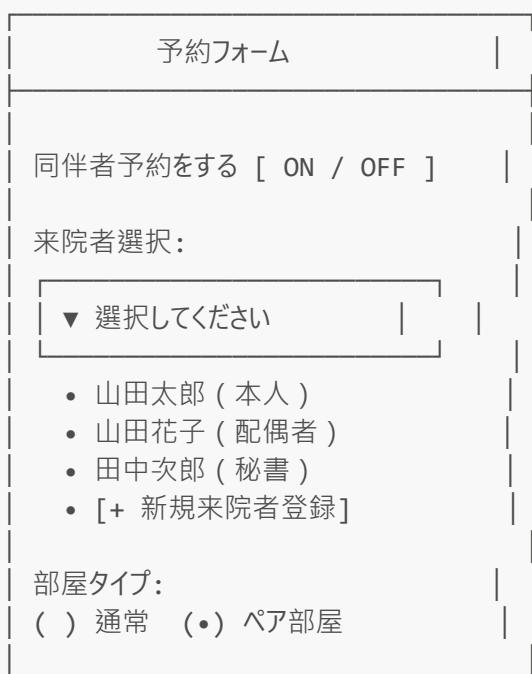
- **本会員**: すべての機能にアクセス可能、チケット使用履歴閲覧可能
- **サブ会員 (秘書)** : 予約機能のみ、チケット残数のみ閲覧可能
- **同伴者**: システムアクセス不可 (代理予約のみ)

## 3. 画面詳細設計

### 3.1 メインメニュー画面



### 3.2 予約フォーム画面



[ 次へ進む ]

### 3.3 ペア部屋予約画面

ペア部屋予約

1人目	2人目
来院者: 山田太郎	来院者: 山田花子
施術メニュー:	施術メニュー:
<input type="checkbox"/> フェイシャル60分	<input type="checkbox"/> フェイシャル60分
<input type="checkbox"/> ボディケア90分	<input type="checkbox"/> ボディケア90分
<input type="checkbox"/> 点滴30分	<input type="checkbox"/> 点滴30分

予約可能日カレンダー  
※ペア部屋が予約可能な日のみ表示

< 2025年1月 >

日	月	火	水	木	金	土
-	-	1	2	3	4	
5	6	7	8	9	10	11
12	13	14	15	16	17	18

時間選択: [10:00] [14:00] [16:00]

[ 予約確認へ ]

### 3.4 通常予約画面

通常予約

来院者: 山田太郎

施術メニュー (複数選択可) :

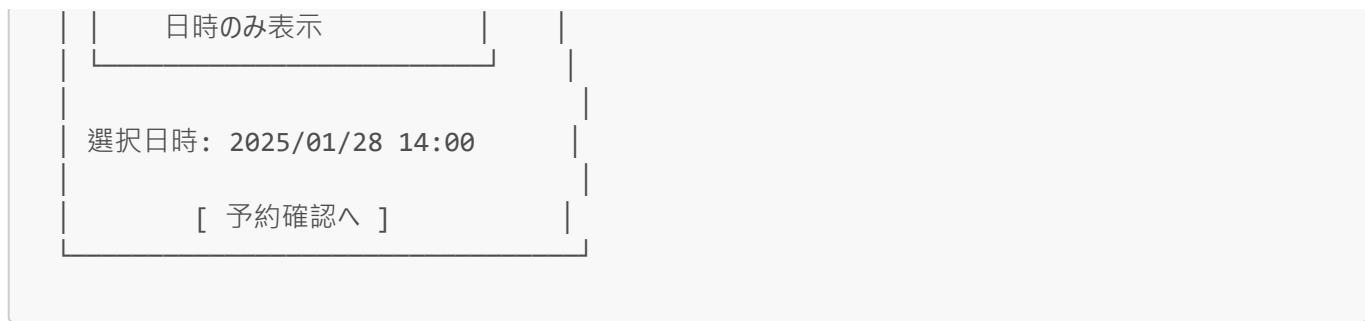
フェイシャル60分

ボディケア90分

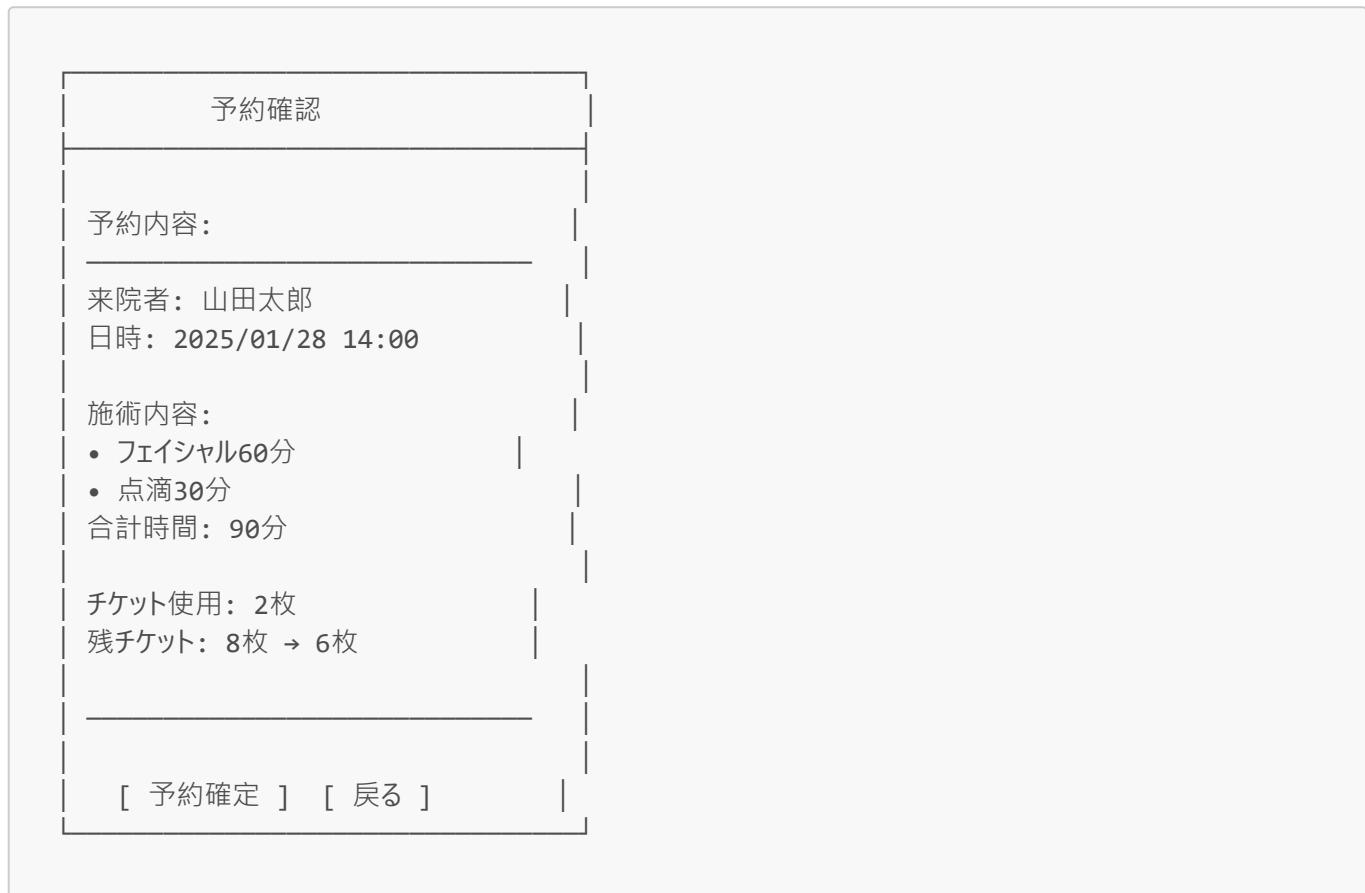
点滴30分

予約可能日時:

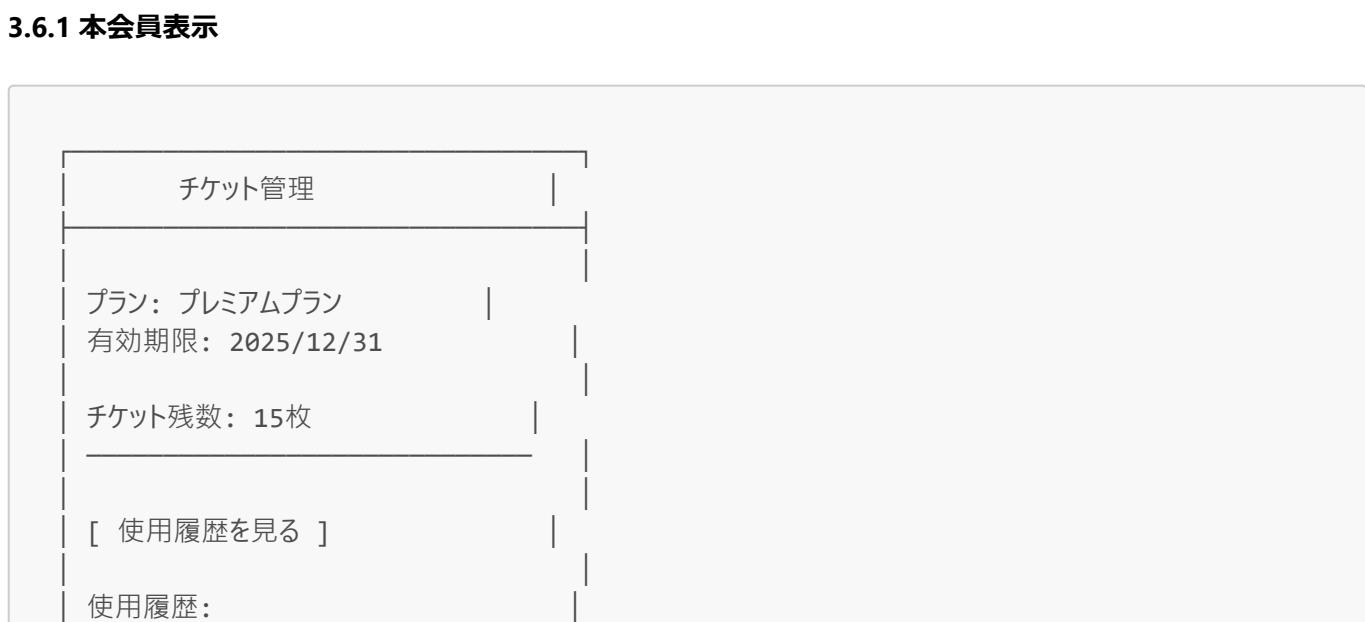
2025年1月カレンダー  
※選択メニューで可能な



### 3.5 予約確認画面



### 3.6 チケット管理画面



2025/01/20 山田太郎  
フェイシャル60分 (1枚)

2025/01/15 山田花子  
ボディケア90分 (2枚)

2025/01/10 田中次郎  
点滴30分 (1枚)

### 3.6.2 サブ会員表示

チケット管理

プラン: プレミアムプラン  
有効期限: 2025/12/31

チケット残数: 15枚

※詳細は本会員様にお問い合わせ  
ください

## 3.7 書籍一覧画面

書籍・資料一覧

施術関連資料:

□ 施術同意書  
[ダウンロード]

□ フェイシャル説明書  
[ダウンロード]

□ アフターケアガイド  
[ダウンロード]

※資料はクリニックスタッフが  
定期的に更新しています

## 4. データフローとAPI連携

### 4.1 予約作成フロー

```

sequenceDiagram
    participant UI as 画面
    participant GAS as Google Apps Script
    participant MF as Medical Force API
    participant Sheet as スプレッドシート

    UI->>GAS: 予約リクエスト
    GAS->>MF: 空き状況確認API
    MF-->>GAS: 空き情報
    GAS->>UI: カレンダー表示

    UI->>GAS: 予約確定
    GAS->>MF: 予約作成API
    MF-->>GAS: 予約ID
    GAS->>Sheet: チケット更新
    GAS-->>UI: 完了通知
  
```

### 4.2 必要なAPI呼び出し

画面	MF APIエンドポイント	用途	必要なデータ
LINE認証	なし (GAS内部処理)	LINE IDから法人グループ特定	LINE ID → visitor_id変換
メインメニュー	GET /developer/clinics	クリニック営業情報取得	営業時間、休診日
予約フォーム	GET /developer/visitors	法人グループの来院者一覧取得	visitor_id, name, code
予約フォーム	GET /developer/menus	施術メニュー一覧取得	menu_id, name, needed_time
ペア部屋予約	POST /developer/vacancies	ペア部屋の空き時間検索	2人分の同時刻空き確認
通常予約	POST /developer/vacancies	通常予約の空き時間検索	選択メニューの空き確認
予約確認	POST /developer/reservations	予約作成	予約ID返却
チケット管理	スプレッドシート	チケット残数取得	残数、有効期限

画面	MF APIエンドポイント	用途	必要なデータ
チケット管理	GET /developer/reservations	使用履歴取得（本会員のみ）	予約履歴からチケット使用算出
書籍一覧	スプレッドシート	資料リンク一覧取得	資料名、URL

## 4.3 画面別API詳細

### 4.3.1 LINE認証・初期画面

#### 処理フロー

1. LINE Webhook経由でLINE IDを取得
2. スプレッドシートでLINE ID → visitor\_id変換
3. visitor\_idからクリニック情報を取得

#### GET /developer/clinics

```
// Headers
{
  "Authorization": "Bearer YOUR_TOKEN",
  "clinic_id": "12345678-0000-0000-0000-000000000000"
}

// Response
{
  "clinic_id": "12345678-0000-0000-0000-000000000000",
  "name": "CultiReFINEクリニック",
  "business_hours": [
    {
      "open_at": "10:00",
      "close_at": "20:00"
    }
  ],
  "closed_days": [
    {
      "date": "2025-01-01"
    }
  ],
  "unit_time_minutes": "30",
  "reservation_span": "90"
}
```

### 4.3.2 予約フォーム画面

#### 画面表示に必要なデータ取得

1. 来院者一覧取得（法人グループ）

```
// GAS側の処理 : LINE IDから法人グループIDを特定後、関連する来院者を取得
const groupVisitors = await getVisitorsByGroupId(groupId);

// 画面に渡すデータ形式
{
  "visitors": [
    {
      "id": "visitor_001",
      "name": "山田太郎",
      "type": "本会員",
      "canBook": true
    },
    {
      "id": "visitor_002",
      "name": "山田花子",
      "type": "同伴者",
      "canBook": false // 代理予約のみ
    }
  ],
  "canAddNewVisitor": true // 本会員/サブ会員のみtrue
}
```

## 2. メニュー一覧取得

```
// GET /developer/menus
// Response
{
  "items": [
    {
      "id": "menu_001",
      "name": "フェイシャル60分",
      "description": "基本的なフェイシャルケア",
      "needed_time": "60",
      "is_pair_room": false
    },
    {
      "id": "menu_pair_001",
      "name": "ペアフェイシャル60分",
      "description": "ペア部屋専用",
      "needed_time": "60",
      "is_pair_room": true
    }
  ],
  "count": "10"
}
```

### 4.3.3 ペア部屋予約画面

#### 空き時間検索（2人分同時確認）

```

// POST /developer/vacancies
// Request
{
  "epoch_from_keydate": "2025-01-28",
  "epoch_to_keydate": "2025-01-31",
  "time_spacing": "30",
  "menus": [
    {
      "menu_id": "menu_pair_001", // 1人目のメニュー
      "staff_ids": []
    },
    {
      "menu_id": "menu_pair_002", // 2人目のメニュー
      "staff_ids": []
    }
  ]
}

// Response
{
  "2025-01-28": {
    "10:00": "ok",
    "10:30": "ok",
    "11:00": "ng", // どちらか一方でもNGならペア予約不可
    "14:00": "ok",
    "16:00": "ok"
  },
  "2025-01-29": {
    "10:00": "ng",
    "14:00": "ok"
  }
}

```

## 画面表示用データ変換

```

// GAS側でペア予約可能な枠のみフィルタリング
function filterPairAvailableSlots(vacancyResponse) {
  const pairSlots = {};

  Object.keys(vacancyResponse).forEach(date => {
    pairSlots[date] = [];
    Object.keys(vacancyResponse[date]).forEach(time => {
      if (vacancyResponse[date][time] === "ok") {
        pairSlots[date].push(time);
      }
    });
  });

  return pairSlots;
}

```

```
// 画面に渡すデータ
{
  "availableDates": {
    "2025-01-28": ["10:00", "10:30", "14:00", "16:00"],
    "2025-01-29": ["14:00"]
  }
}
```

#### 4.3.4 通常予約画面

##### 複数メニュー選択時の空き時間検索

```
// POST /developer/vacancies
// Request ( 複数メニュー選択時 )
{
  "epoch_from_keydate": "2025-01-28",
  "epoch_to_keydate": "2025-01-31",
  "time_spacing": "30",
  "menus": [
    {
      "menu_id": "menu_001", // フェイシャル60分
      "staff_ids": []
    },
    {
      "menu_id": "menu_003", // 点滴30分
      "staff_ids": []
    }
  ]
}

// Response
{
  "2025-01-28": {
    "10:00": "ok",
    "10:30": "ng", // 合計90分必要なので、この時間は不可
    "11:00": "ok",
    "14:00": "ok"
  }
}
```

##### 画面表示用データ

```
{
  "selectedMenus": [
    {
      "id": "menu_001",
      "name": "フェイシャル60分",
      "time": 60
    }
  ]
}
```

```

},
{
  "id": "menu_003",
  "name": "点滴30分",
  "time": 30
}
],
"totalTime": 90,
"availableSlots": {
  "2025-01-28": ["10:00", "11:00", "14:00"],
  "2025-01-29": ["10:00", "14:00", "16:00"]
}
}
}

```

#### 4.3.5 予約確認・作成画面

##### 予約作成API呼び出し

###### 1. 通常予約の作成

```

// POST /developer/reservations
// Request
{
  "visitor_id": "visitor_001",
  "start_at": "2025-01-28T14:00:00.000+09:00",
  "note": "LINE予約",
  "is_online": false,
  "menus": [
    {
      "menu_id": "menu_001"
    },
    {
      "menu_id": "menu_003"
    }
  ],
  "api_collector_id": "LINE_BOT",
  "api_collector_reservation_id": "line_res_20250128_001"
}

// Response
{
  "reservation_id": "res_12345678"
}

```

###### 2. ペア部屋予約の作成（2回実行）

```

// GAS側の実装
async function createPairReservation(pairData) {
  const timestamp = Date.now();

```

```

const reservations = [];

try {
  // 1人目の予約
  const res1 = await createReservation({
    visitor_id: pairData.visitor1.id,
    start_at: pairData.startAt,
    note: "ペア部屋予約 (1人目)",
    menus: [{ menu_id: pairData.menu1 }],
    api_collector_id: "LINE_BOT",
    api_collector_reservation_id: `pair_${timestamp}_1`
  });
  reservations.push(res1);

  // 2人目の予約
  const res2 = await createReservation({
    visitor_id: pairData.visitor2.id,
    start_at: pairData.startAt,
    note: "ペア部屋予約 (2人目)",
    menus: [{ menu_id: pairData.menu2 }],
    api_collector_id: "LINE_BOT",
    api_collector_reservation_id: `pair_${timestamp}_2`
  });
  reservations.push(res2);

  return {
    success: true,
    reservationIds: [res1.reservation_id, res2.reservation_id]
  };
}

} catch (error) {
  // ロールバック処理
  for (const res of reservations) {
    await cancelReservation(res.reservation_id);
  }
  throw error;
}
}
}

```

#### 4.3.6 チケット管理画面

##### チケット情報と使用履歴の取得

###### 1. チケット残数 (スプレッドシートから)

```

// GAS側でスプレッドシートから取得
{
  "ticketInfo": {
    "planName": "プレミアムプラン",
    "totalTickets": 20,
    "remainingTickets": 15,
  }
}

```

```

        "validUntil": "2025-12-31",
        "allowedMenus": ["全施術"]
    }
}
}

```

## 2. 使用履歴（本会員のみ）

```

// GET /developer/reservations
// Request parameters
{
  "visitor_id": "group_visitors", // 法人グループ全体
  "epoch_from": "2025-01-01T00:00:00+09:00",
  "epoch_to": "2025-01-31T23:59:59+09:00"
}

// Response ( 予約情報から使用チケット数を算出 )
{
  "items": [
    {
      "id": "res_001",
      "start_at": "2025-01-20T14:00:00.000+09:00",
      "visitor": {
        "name": "山田太郎"
      },
      "menus": [
        {
          "name": "フェイシャル60分"
        }
      ],
      "ticket_used": 1 // GAS側で計算
    }
  ]
}

```

## 画面表示データ（権限による制御）

```

// 本会員向け
{
  "memberType": "MAIN",
  "ticketInfo": { /* 上記と同じ */ },
  "usageHistory": [
    {
      "date": "2025/01/20",
      "visitorName": "山田太郎",
      "menuName": "フェイシャル60分",
      "ticketsUsed": 1
    },
    {
      "date": "2025/01/15",

```

```
        "visitorName": "山田花子",
        "menuName": "ボディケア90分",
        "ticketsUsed": 2
    }
]
}

// サブ会員向け
{
    "memberType": "SUB",
    "ticketInfo": {
        "remainingTickets": 15,
        "validUntil": "2025-12-31"
    },
    "usageHistory": null // 表示しない
}
```

#### 4.3.7 新規来院者登録

##### 来院者登録API

```
// POST /developer/visitors
// Request
{
    "first_name": "次郎",
    "last_name": "田中",
    "first_name_kana": "ジロウ",
    "last_name_kana": "タナカ",
    "gender": "MALE",
    "email": "tanaka@example.com",
    "code": "AUTO_GENERATED_001", // GAS側で自動生成
    "birthday": "1990-05-15",
    "api_collector_id": "LINE_BOT",
    "api_collector_customer_id": "line_tanaka_001",
    "note": {
        "type": "doc",
        "content": [
            {
                "type": "paragraph",
                "content": [
                    {
                        "type": "text",
                        "text": "同伴者（山田太郎様の紹介）"
                    }
                ]
            }
        ]
    }
}

// Response
{
    "id": "visitor_new_001",
    "name": "田中次郎",
```

```
    "code": "AUTO_GENERATED_001"  
}
```

## 登録後の処理

```
// GAS側で法人グループに紐付け  
await linkVisitorToGroup({  
  visitorId: "visitor_new_001",  
  groupId: currentGroupId,  
  relationship: "同伴者",  
  addedBy: currentUserId  
});
```

## 4.4 エラーハンドリングとキャッシュ戦略

### 4.4.1 APIエラーハンドリング

#### 共通エラー処理

```
// GAS側のエラーハンドリング  
function handleApiError(error, screen) {  
  const errorMessages = {  
    401: {  
      message: "認証エラーが発生しました。再度ログインしてください。",  
      action: "reauth"  
    },  
    400: {  
      // 画面別のエラーメッセージ  
      reservation: {  
        "Menu not found": "選択されたメニューが見つかりません。",  
        "Visitor not found": "来院者情報が見つかりません。",  
        "dates_filled": "選択された時間帯は既に予約で埋まっています."  
      }  
    },  
    429: {  
      message: "アクセスが集中しています。しばらくお待ちください。",  
      action: "retry"  
    },  
    500: {  
      message: "システムエラーが発生しました。管理者にお問い合わせください。",  
      action: "contact"  
    }  
  };  
  
  return {  
    status: "error",  
    code: error.status,  
    message: errorMessages[error.status]?.message || "エラーが発生しました。",  
    userAction: errorMessages[error.status]?.action,  
  };  
}
```

```

        details: error.details
    };
}

```

## 画面別エラー表示

```

// 予約作成時のエラー
{
  "status": "error",
  "errorType": "SLOT_UNAVAILABLE",
  "message": "選択された時間帯は既に予約されています。",
  "suggestions": [
    "2025-01-28 15:00",
    "2025-01-28 16:00",
    "2025-01-29 14:00"
  ]
}

// ペア部屋予約のロールバック通知
{
  "status": "error",
  "errorType": "PAIR_BOOKING_FAILED",
  "message": "ペア部屋の予約に失敗しました。両方の予約をキャンセルしました。",
  "details": {
    "visitor1": "予約成功",
    "visitor2": "予約失敗 ( 時間重複 )"
  }
}

```

## 4.4.2 キャッシュ戦略

### キャッシュ実装

```

// GAS側のキャッシュ管理
const CacheManager = {
  // キャッシュ時間設定 ( 秒 )
  TTL: {
    CLINIC_INFO: 86400,      // 24時間
    MENU_LIST: 86400,        // 24時間
    VISITOR_INFO: 1800,      // 30分
    VACANCIES: 0,            // キャッシュなし (リアルタイム性重視 )
    TICKET_INFO: 300         // 5分
  },
  // キャッシュ取得
  get(key) {
    const cache = CacheService.getScriptCache();
    const data = cache.get(key);
    return data ? JSON.parse(data) : null;
  }
}

```

```
},  
  
    // キャッシュ設定  
    set(key, data, ttl) {  
        const cache = CacheService.getScriptCache();  
        cache.put(key, JSON.stringify(data), ttl);  
    },  
  
    // キャッシュクリア  
    clear(pattern) {  
        // 特定パターンのキャッシュをクリア  
        const cache = CacheService.getScriptCache();  
        if (pattern === 'VISITOR_*') {  
            // 来院者関連のキャッシュをクリア  
        }  
    }  
};
```

## 画面別キャッシュ利用

```
// メニュー一覧取得（キャッシュ利用）  
async function getMenusWithCache() {  
    const cacheKey = 'MENU_LIST_ALL';  
  
    // キャッシュチェック  
    const cached = CacheManager.get(cacheKey);  
    if (cached) {  
        return cached;  
    }  
  
    // APIコール  
    const menus = await MFApi.getMenus();  
  
    // キャッシュ保存  
    CacheManager.set(cacheKey, menus, CacheManager.TTL.MENU_LIST);  
  
    return menus;  
}  
  
// 空き時間検索（キャッシュなし）  
async function getVacancies(params) {  
    // 常に最新情報を取得  
    return await MFApi.getVacancies(params);  
}
```

## 4.5 データ同期とバッチ処理

### 定期実行タスク

```
// 1時間ごとのチケット情報更新
function syncTicketInfo() {
  const groups = getActiveGroups();

  groups.forEach(group => {
    try {
      // 予約履歴から使用チケット数を計算
      const reservations = MFApi.getReservations({
        visitor_id: group.visitors,
        epoch_from: getMonthStart(),
        epoch_to: getMonthEnd()
      });

      // チケット使用数を更新
      updateTicketUsage(group.id, calculateTicketUsage(reservations));

    } catch (error) {
      console.error(`Ticket sync failed for group ${group.id}:`, error);
    }
  });
}

// 日次でのデータ整合性チェック
function dailyDataValidation() {
  // 1. 予約とチケットの整合性
  // 2. 来院者情報の更新チェック
  // 3. 期限切れデータのクリーンアップ
}
```

## 5. UI/UX考慮事項

### 5.1 レスポンシブデザイン

- モバイルファースト設計 (LINE内ブラウザ最適化)
- タブレット対応 (スタッフ利用想定)

### 5.2 エラーハンドリング

- API通信エラー時の再試行ボタン
- 予約枠なしの場合の代替日時提案
- チケット不足時の明確なメッセージ

### 5.3 ローディング状態

- API通信中のスピナー表示
- 段階的な画面表示 (スケルトンスクリーン)

### 5.4 アクセシビリティ

- 大きめのタップ領域 (44px以上)

- 高コントラストな配色
- 読みやすいフォントサイズ (16px以上)

## 6. セキュリティ考慮事項

### 6.1 表示制御

- サブ会員には他のサブ会員の予約を表示しない
- チケット使用履歴は本会員のみ閲覧可能
- 同伴者情報の適切なマスキング

### 6.2 データ保護

- LINE IDによる認証 (セッション不要)
- 個人情報の最小限表示
- ログへの個人情報出力禁止

---

バージョン: 2.0

最終更新: 2025年1月28日

更新内容: MF API仕様に基づく詳細なAPI連携とJSONデータ形式を追加