Oz Kitchen – Subscription Meal Mini■App (Web/Telegram)

Oz Kitchen – Subscription Meal Mini■App (Web/Telegram)

Full Developer Documentation • Generated on 2025-10-15 12:57

1. Project Overview

Oz Kitchen is a subscription-based lunch & dinner delivery platform. Originally targeted at college students, it is now expanding for general customers. This repository contains a React + TypeScript Vite app designed as a lightweight web app and as a Telegram Mini App. The current demo implements core UX flows (pricing → calendar selection → packaging → payment mock).

2. Scope & Goals

• Offer weekly/monthly meal subscriptions with lunch and/or dinner on working days.

• Allow users to choose delivery days from a calendar and set packaging preferences.

• Support Telegram Mini App launch and standard web deployment.

• Integrate Supabase for user auth, data persistence, and admin operations (roadmap).

3. System Architecture

Frontend: React 18 + TypeScript + Vite, TailwindCSS, lucide-react icons. Integrations: @telegram-apps/sdk for Telegram Mini App; @supabase/supabase-js (planned/partial). Backend: Supabase (Postgres + Auth + Storage) — to be connected for production. Build & Dev: Vite scripts (dev, build, preview).

3.1 Key Dependencies (package.json)

```
{
  "dependencies": {
    "@supabase/supabase-js": "^2.57.4",
    "@telegram-apps/sdk": "^3.11.5",
    "lucide-react": "^0.344.0",
    "react": "^18.3.1",
    "react-dom": "^18.3.1"
  },
  "devDependencies": {
    "@eslint/js": "^9.9.1",
    "@types/react": "^18.3.5",
    "@types/react-dom": "^18.3.0",
    "@vitejs/plugin-react": "^4.3.1",
    "autoprefixer": "^10.4.18",
    "eslint": "^9.9.1",
    "eslint-plugin-react-hooks": "^5.1.0-rc.0",
    "eslint-plugin-react-refresh": "^0.4.11",
    "globals": "^15.9.0",
    "postcss": "^8.4.35",
    "tailwindcss": "^3.4.1",
    "typescript": "^5.5.3",
    "typescript-eslint": "^8.3.0",
    "vite": "^5.4.2"
  }
}
```

4. Setup & Local Development

• Node.js 18+ and pnpm/npm/yarn installed.

• Clone the repository and enter the 'project' directory.

• Install dependencies: `npm install`

• Start dev server: `npm run dev` (Vite)

• Preview production build: `npm run build && npm run preview`

Environment variables (create `.env` at project root):
VITE_SUPABASE_URL=<your_supabase_url>
VITE_SUPABASE_ANON_KEY=<your_supabase_anon_key>
VITE_TELEGRAM_BOT_NAME=<your_bot_username>
VITE_TELEGRAM_WEBAPP_URL=<deployed_webapp_url>

5. Repository Structure (truncated)

ozkitchen_project/
■■■ project
■■■ .bolt
■ ■■■ config.json
■ ■■■ prompt
■■■ src
■ ■■■ components
■ ■ ■■■ CalendarScreen.tsx
■ ■ ■■■ PackagingScreen.tsx
■ ■ ■■■ PaymentScreen.tsx
■ ■ ■■■ PricingScreen.tsx
■ ■■■ data
■ ■ ■■■ mockData.ts
■ ■■■ lib
■ ■ ■■■ telegram.ts
■ ■■■ types
■ ■ ■■■ index.ts
■ ■■■ App.tsx
■ ■■■ index.css
■ ■■■ main.tsx
■ ■■■ vite-env.d.ts
■■■ .gitignore
■■■ eslint.config.js
■■■ index.html
■■■ package-lock.json
■■■ package.json
■■■ postcss.config.js
■■■ tailwind.config.js
■■■ tsconfig.app.json
■■■ tsconfig.json
■■■ tsconfig.node.json
■■■ vite.config.ts

6. Frontend Modules & Screens

This demo uses a simple screen flow controlled in App.tsx.

• PricingScreen.tsx – choose plan (weekly/monthly), meals/day, and price overview.

• CalendarScreen.tsx – select working days for lunch/dinner deliveries.

• PackagingScreen.tsx – choose packaging preferences (e.g., reusable vs disposable).

• PaymentScreen.tsx – mock checkout summary (to be wired to real payments).

• lib/telegram.ts – Telegram Mini App bootstrap (init data, theme, BackButton/MainButton).

• data/mockData.ts – mock menu/plans for demo.

6.1 Notable Files (Excerpts)

App Entrypoint: src/App.tsx

```
import { useState, useEffect } from 'react';
import PricingScreen from './components/PricingScreen';
import PackagingScreen from './components/PackagingScreen';
import CalendarScreen from './components/CalendarScreen';
import PaymentScreen from './components/PaymentScreen';
import { SubscriptionPlan, OrderItem } from './types';
import { subscriptionPlans, menuItems } from './data/mockData';
import { initTelegram } from './lib/telegram';

type Screen = 'pricing' | 'packaging' | 'calendar' | 'payment' | 'complete';

function App() {
  const [currentScreen, setCurrentScreen] = useState<Screen>('pricing');
  const [selectedPlan, setSelectedPlan] = useState<SubscriptionPlan | null>(null);
  const [customPrice, setCustomPrice] = useState<number>(0);
  const [orderItems, setOrderItems] = useState<OrderItem[]>([]);
  const [deliveryDate, setDeliveryDate] = useState<Date | null>(null);
  const [paymentMethod, setPaymentMethod] = useState<string>('');

  useEffect(() => {
    const telegram = initTelegram();

    if (telegram.backButton) {
      telegram.backButton.onClick(() => {
        handleBack();
      });
    }

    if (telegram.mainButton) {
      telegram.mainButton.setParams({
```

Telegram Mini App Integration: src/lib/telegram.ts

```
import { init, backButton, mainButton, themeParams } from '@telegram-apps/sdk';

export const initTelegram = () => {
  try {
    init();

    if (backButton.isSupported()) {
      backButton.mount();
    }

    if (mainButton.isSupported()) {
      mainButton.mount();
    }
```

```
    return {
      backButton: backButton.isSupported() ? backButton : null,
      mainButton: mainButton.isSupported() ? mainButton : null,
      themeParams: themeParams.isSupported() ? themeParams : null,
    };
  } catch (error) {
    console.warn('Telegram WebApp SDK not available', error);
    return {
      backButton: null,
      mainButton: null,
      themeParams: null,
    };
  }
};
```

Pricing Screen: src/components/PricingScreen.tsx

```
import { useState } from 'react';
import { SubscriptionPlan } from '../types';
import { ArrowRight } from 'lucide-react';

interface PricingScreenProps {
  plans: SubscriptionPlan[];
  onNext: (plan: SubscriptionPlan, customPrice: number) => void;
}

export default function PricingScreen({ plans, onNext }: PricingScreenProps) {
  const [selectedPlan, setSelectedPlan] = useState<string>('monthly');
  const [customPrice, setCustomPrice] = useState<string>('');

  const currentPlan = plans.find(p => p.id === selectedPlan);

  const handleNext = () => {
    if (currentPlan && customPrice) {
      const price = parseFloat(customPrice);
      if (price >= currentPlan.priceRangeStart && price <= currentPlan.priceRangeEnd) {
        onNext(currentPlan, price);
      }
    }
  };

  return (
    <div className="min-h-screen bg-[#1a4d4d] text-white flex flex-col">
      <div className="flex-1 flex flex-col p-6">
        <div className="flex items-center gap-3 mb-8">
          <div className="w-12 h-12 bg-[#f59e42] rounded-full flex items-center justify-
center">
            <span className="text-2xl">■</span>
          </div>
          <div>
            <h1 className="text-2xl font-bold">OZ</
```

Calendar Screen: src/components/CalendarScreen.tsx

```
import { useState } from 'react';
import { ChevronLeft, ChevronRight } from 'lucide-react';
import { OrderItem } from '../types';

interface CalendarScreenProps {
  orderItems: OrderItem[];
  onNext: (date: Date) => void;
  onBack: () => void;
}

export default function CalendarScreen({ orderItems, onNext, onBack }: CalendarScreenProps) {
  const [currentDate, setCurrentDate] = useState(new Date());
  const [selectedDate, setSelectedDate] = useState<Date | null>(null);

  const getDaysInMonth = (date: Date) => {
    const year = date.getFullYear();
    const month = date.getMonth();
    const firstDay = new Date(year, month, 1);
    const lastDay = new Date(year, month + 1, 0);
    const daysInMonth = lastDay.getDate();
    const startingDayOfWeek = firstDay.getDay();

    return { daysInMonth, startingDayOfWeek };
  };

  const { daysInMonth, startingDayOfWeek } = getDaysInMonth(currentDate);

  const monthNames = [
    'January', 'February', 'March', 'April', 'May', 'June',
    'July', 'August', 'September', 'October', 'November', 'December'
  ];

  const weekDays = ['SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT'];

  const changeMonth = (delta: number) => {
    setCurrentDate(p
```

Packaging Screen: src/components/PackagingScreen.tsx

```
import { useState } from 'react';
import { MenuItem, OrderItem } from '../types';
import { ChevronLeft, Minus, Plus } from 'lucide-react';

interface PackagingScreenProps {
  menuItems: MenuItem[];
  onNext: (items: OrderItem[]) => void;
  onBack: () => void;
}

export default function PackagingScreen({ menuItems, onNext, onBack }: PackagingScreenProps) {
  const [selectedItems, setSelectedItems] = useState<Map<string, number>>(new Map());

  const updateQuantity = (itemId: string, delta: number) => {
    setSelectedItems(prev => {
```

```
      const newMap = new Map(prev);
      const currentQty = newMap.get(itemId) || 0;
      const newQty = Math.max(0, currentQty + delta);

      if (newQty === 0) {
        newMap.delete(itemId);
      } else {
        newMap.set(itemId, newQty);
      }

      return newMap;
    });
  };

  const getTotalPrice = () => {
    return Array.from(selectedItems.entries()).reduce((total, [itemId, qty]) => {
      const item = menuItems.find(m => m.id === itemId);
      return total + (item?.price || 0) * qty;
    }, 0);
  };

  const getTotalItems = () => {
    return Array.from(selectedItems.values()).reduce((sum, qty) => sum + qty, 0);
  };

  const handle
```

Payment Screen: src/components/PaymentScreen.tsx

```
import { useState } from 'react';
import { ChevronLeft } from 'lucide-react';
import { OrderItem } from '../types';

interface PaymentScreenProps {
  orderItems: OrderItem[];
  deliveryDate: Date;
  onComplete: (paymentMethod: string) => void;
  onBack: () => void;
}

export default function PaymentScreen({ orderItems, deliveryDate, onComplete, onBack }:
PaymentScreenProps) {
  const [selectedPayment, setSelectedPayment] = useState<string>('');

  const getTotalPrice = () => {
    return orderItems.reduce((sum, item) => sum + item.price * item.quantity, 0);
  };

  const formatDate = (date: Date) => {
    const options: Intl.DateTimeFormatOptions = {
      weekday: 'short',
      year: 'numeric',
      month: 'short',
      day: 'numeric'
```

```
    };
    return date.toLocaleDateString('en-US', options);
  };

  const handleComplete = () => {
    if (selectedPayment) {
      onComplete(selectedPayment);
    }
  };

  return (
    <div className="min-h-screen bg-gradient-to-b from-[#2a5a5a] to-[#1a4d4d] text-white flex
flex-col">
      <div className="flex-1 flex flex-col">
        <div className="p-6 pb-4">
          <div className="flex items-center gap-4 mb-6">
            <button
```

7. State & Navigation

The app stores minimal UI state in React hooks at the App level and passes props to child screens. For production, consider a state library (Zustand or Redux Toolkit) and URL-based routing (React Router) to support deep links and multi-step flows.

8. Supabase Integration Plan

• Auth: Email OTP (magic link) and Telegram OAuth (via WebApp initData) mapping to users table.

• Core tables: users, addresses, plans, subscriptions, orders, order_items, delivery_slots, payments, menu_items, packaging_preferences.

• RLS policies: row-level security by user_id; service role for admin tasks.

• Edge Functions (optional): create subscription, schedule deliveries, handle payment webhooks.

• Storage: menu images, receipts.

Tables (proposed):
- users(id uuid pk, phone text, email text, telegram_id text unique, full_name text, created_at timestamptz)
- addresses(id uuid pk, user_id uuid fk, line1 text, area text, city text, notes text)
- plans(id uuid pk, name text, period text check in ('weekly','monthly'), meals_per_day int, price_cents int)
- subscriptions(id uuid pk, user_id uuid fk, plan_id uuid fk, start_date date, end_date date, status text check in ('active','paused','canceled'))
- delivery_days(id uuid pk, subscription_id uuid fk, date date, lunch boolean, dinner boolean)
- packaging_preferences(id uuid pk, user_id uuid fk, type text check in ('reusable','disposable'), notes text)
- orders(id uuid pk, subscription_id uuid fk, scheduled_for date, slot text check in ('lunch','dinner'), status text)
- order_items(id uuid pk, order_id uuid fk, menu_item_id uuid fk, qty int)
- menu_items(id uuid pk, name text, description text, kcal int, price_cents int, image_url text)
- payments(id uuid pk, subscription_id uuid fk, provider text, amount_cents int, currency text, status text, external_ref text)

9. API Design (Optional Backend)

While Supabase provides direct database access, a thin API layer can encapsulate business logic. Endpoints (example): POST /auth/telegram/link, POST /subscriptions, GET /subscriptions/:id, POST /orders/schedule, POST /payments/checkout (creates payment intent). Use JWT from Supabase auth.

10. Payments

• Stripe (international) or local PSPs (e.g., Chapa/Telebirr in Ethiopia).

• Use Payment Links or client→server intents. Store webhooks → update payments & subscriptions.

• Telegram Mini App: openInvoice or external checkout in WebApp. Ensure SCA and receipt emails.

11. Telegram Mini App Deployment

• Create a bot with @BotFather, enable Web App with web_app URL.

• Host the Vite build (e.g., Vercel/Netlify/Supabase Hosting).

• In lib/telegram.ts, initialize Telegram SDK and handle theme/MainButton events.

• Validate initData hash on server when linking Telegram user → Supabase user.

12. Security & Compliance

• Enforce RLS on Supabase tables; never expose service role client-side.

• Validate Telegram initData via HMAC per Telegram docs.

• Protect secrets via .env and build-time VITE_ variables.

• Log PII access, encrypt at rest via Postgres + TLS; minimize stored PII.

• Comply with local data/consumer laws for subscriptions & recurring billing.

13. CI/CD & Deployment

• Use GitHub Actions: lint, typecheck, build on PR.

• Preview deploys to Vercel/Netlify; production on tag release.

• Run Supabase migrations via CLI in release pipeline (if used).

14. Roadmap

• Hook real Supabase auth and profiles; replace mockData with DB.

• Add address management and delivery zones for public users.

• Admin dashboard for menu, schedules, and delivery ops.

• Notifications (Telegram push & SMS) for delivery reminders.

• Offline-friendly mobile PWA with add-to-home-screen.

• Analytics: subscription retention, on-time delivery rate, meal ratings.

15. Appendix – Source Files in src/

- src/App.tsx

- src/components/CalendarScreen.tsx

- src/components/PackagingScreen.tsx

- src/components/PaymentScreen.tsx

- src/components/PricingScreen.tsx

- src/data/mockData.ts

- src/index.css
- src/lib/telegram.ts
- src/main.tsx
- src/types/index.ts
- src/vite-env.d.ts