Diet Planner — Fullstack Starter (Frontend + Backend + DB)

This document contains a **ready-to-copy** starter project: backend (FastAPI + MongoDB) and frontend (React + Tailwind) plus Docker, `docker-compose.yml`, and database management notes. Copy each file into your project folder exactly as named.

Project tree

```
diet-planner/
├─ backend/
│  ├─ app/
│  │  ├─ main.py
│  │  ├─ api/
│  │  │  ├─ auth.py
│  │  │  ├─ answers.py
│  │  │  ├─ diet.py
│  │  │  ├─ chatbot.py
│  │  │  └─ progress.py
│  │  ├─ core/
│  │  │  ├─ config.py
│  │  │  ├─ security.py
│  │  │  └─ ai_client.py
│  │  ├─ db/
│  │  │  └─ mongo.py
│  │  ├─ models/
│  │  │  ├─ user_models.py
│  │  │  └─ diet_models.py
│  │  └─ utils/
│  │     └─ pdf_helper.py
│  ├─ requirements.txt
│  ├─ Dockerfile
│  └─ .env.sample
├─ frontend/
│  ├─ public/
│  │  └─ index.html
│  ├─ src/
│  │  ├─ index.jsx
│  │  ├─ App.jsx
│  │  ├─ i18n.js
│  │  ├─ api/
│  │  │  └─ apiClient.js
│  │  ├─ pages/
│  │  │  ├─ Login.jsx
│  │  │  ├─ Signup.jsx
│   │   │   ├─ Dashboard.jsx
│   │   │   ├─ AnalysisForm.jsx
│   │   │   └─ Profile.jsx
│   │   ├─ components/
│   │   │   ├─ NavMenu.jsx
│   │   │   ├─ DietTable.jsx
│   │   │   ├─ ChartsPanel.jsx
│   │   │   └─ ChatBot.jsx
│   │   └─ index.css
│   ├─ package.json
│   ├─ tailwind.config.js
│   └─ Dockerfile
├─ docker-compose.yml
└─ README.md
```

BACKEND

backend/requirements.txt

```
fastapi
uvicorn[standard]
motor
python-jose[cryptography]
passlib[bcrypt]
python-dotenv
httpx
pydantic
python-multipart
jinja2
reportlab
```

---

backend/.env.sample

```
MONGO_URI=mongodb://mongodb:27017/dietdb
JWT_SECRET=change_this_to_a_long_random_secret
JWT_ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=60
REFRESH_TOKEN_EXPIRE_DAYS=7
OPENAI_API_KEY=
```

---

backend/Dockerfile

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt /app/
RUN pip install --no-cache-dir -r requirements.txt
COPY ./app /app/app
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000", "--
reload"]
```

---

backend/app/main.py

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.api import auth, answers, diet, chatbot, progress
import os

app = FastAPI(title="Diet Planner API")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(auth.router)
app.include_router(answers.router)
app.include_router(diet.router)
app.include_router(chatbot.router)
app.include_router(progress.router)

@app.get("/")
async def root():
    return {"status": "ok", "service": "diet-planner-backend"}
```

---

backend/app/core/config.py

```
from pydantic import BaseSettings

class Settings(BaseSettings):
    MONGO_URI: str
    JWT_SECRET: str
    JWT_ALGORITHM: str = "HS256"
    ACCESS_TOKEN_EXPIRE_MINUTES: int = 60
    REFRESH_TOKEN_EXPIRE_DAYS: int = 7
    OPENAI_API_KEY: str = None

    class Config:
        env_file = ".env"

settings = Settings()
```

---

backend/app/core/security.py

```
from datetime import datetime, timedelta
from jose import jwt, JWTError
from fastapi import HTTPException, Depends
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
from app.core.config import settings

security = HTTPBearer()

def create_access_token(data: dict, expires_delta: timedelta = None):
    to_encode = data.copy()
    expire = datetime.utcnow() + (expires_delta or
timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES))
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, settings.JWT_SECRET,
algorithm=settings.JWT_ALGORITHM)
    return encoded_jwt

async def verify_token(credentials: HTTPAuthorizationCredentials =
Depends(security)):
    token = credentials.credentials
    try:
        payload = jwt.decode(token, settings.JWT_SECRET,
algorithms=[settings.JWT_ALGORITHM])
        return payload
    except JWTError:
        raise HTTPException(status_code=401, detail="Invalid or expired
token")
```

---

backend/app/db/mongo.py

```
from motor.motor_asyncio import AsyncIOMotorClient
from app.core.config import settings
from bson.objectid import ObjectId
```

```
client = AsyncIOMotorClient(settings.MONGO_URI)
db = client.get_default_database() if client else None

users = db.get_collection("users")
answers = db.get_collection("answers")
diet_plans = db.get_collection("diet_plans")
progress = db.get_collection("progress")

# helper
def objid(id_str):
    return ObjectId(id_str)
```

---

backend/app/models/user_models.py

```
from pydantic import BaseModel, EmailStr, Field
from typing import Optional, Dict, Any

class UserCreate(BaseModel):
    email: EmailStr
    password: str
    name: Optional[str]

class UserPublic(BaseModel):
    id: str = Field(..., alias="_id")
    email: EmailStr
    name: Optional[str]
    age: Optional[int]
    gender: Optional[str]
    height_cm: Optional[float]
    weight_kg: Optional[float]
    preferences: Optional[Dict[str, Any]]
```

---

backend/app/models/diet_models.py

```
from pydantic import BaseModel
from typing import List, Dict, Any

class DietItem(BaseModel):
    name: str
    amount_g: float
    calories: float
    rasa: str
    virya: str
    vipaka: str
```

```
class Meal(BaseModel):
    time: str
    meal_type: str
    items: List[DietItem]

class DietPlan(BaseModel):
    user_id: str
    generated_at: str
    plan: List[Meal]
    summary: Dict[str, Any]
```

---

backend/app/api/auth.py

```
from fastapi import APIRouter, HTTPException
from passlib.context import CryptContext
from pydantic import BaseModel, EmailStr
from app.db.mongo import users
from app.core.security import create_access_token
from datetime import datetime

pwd_ctx = CryptContext(schemes=["bcrypt"], deprecated="auto")
router = APIRouter(prefix="/auth", tags=["auth"])

class SignupModel(BaseModel):
    email: EmailStr
    password: str
    name: str = None

class LoginModel(BaseModel):
    email: EmailStr
    password: str

@router.post("/signup")
async def signup(payload: SignupModel):
    if await users.find_one({"email": payload.email}):
        raise HTTPException(400, "Email already registered")
    pwd_hash = pwd_ctx.hash(payload.password)
    user = {"email": payload.email, "password_hash": pwd_hash, "name":
payload.name,
            "created_at": datetime.utcnow(), "updated_at": datetime.utcnow(),
"preferences": {}}
    res = await users.insert_one(user)
    user_obj = await users.find_one({"_id": res.inserted_id})
    token = create_access_token({"user_id": str(res.inserted_id), "email":
payload.email})
    return {"access_token": token, "token_type": "bearer", "user": {"id":
str(res.inserted_id), "email": payload.email, "name": payload.name}}
```

```
@router.post("/login")
async def login(payload: LoginModel):
    u = await users.find_one({"email": payload.email})
    if not u or not pwd_ctx.verify(payload.password, u["password_hash"]):
        raise HTTPException(401, "Invalid credentials")
    token = create_access_token({"user_id": str(u["_id"]), "email":
u["email"]})
    return {"access_token": token, "token_type": "bearer", "user": {"id":
str(u["_id"]), "email": u["email"], "name": u.get("name")}}
```

---

backend/app/api/answers.py

```
from fastapi import APIRouter, HTTPException, Depends
from pydantic import BaseModel
from datetime import datetime
from app.db.mongo import answers

router = APIRouter(prefix="/answers", tags=["answers"])

class AnswersModel(BaseModel):
    user_id: str
    answers: list

@router.post("/")
async def save_answers(payload: AnswersModel):
    doc = {"user_id": payload.user_id, "answers": payload.answers,
"submitted_at": datetime.utcnow()}
    res = await answers.insert_one(doc)
    doc["_id"] = str(res.inserted_id)
    return doc

@router.get("/{user_id}")
async def get_answers(user_id: str):
    cursor = answers.find({"user_id": user_id}).sort("submitted_at", -1)
    out = []
    async for d in cursor:
        d["_id"] = str(d["_id"])
        out.append(d)
    return out
```

---

backend/app/api/diet.py

```
from fastapi import APIRouter, HTTPException, Depends
from pydantic import BaseModel
from datetime import datetime
from app.db.mongo import users, answers, diet_plans
from app.core.ai_client import generate_diet_plan_ai,
generate_diet_plan_rulebased
from bson.objectid import ObjectId
from app.core.security import verify_token

router = APIRouter(prefix="/diet", tags=["diet"])

class DietReq(BaseModel):
    user_id: str
    answers_id: str

@router.post("/generate")
async def generate(req: DietReq, token = Depends(verify_token)):
    user = await users.find_one({"_id": ObjectId(req.user_id)})
    ans = await answers.find_one({"_id": ObjectId(req.answers_id)})
    if not user or not ans:
        raise HTTPException(404, "User or answers not found")
    try:
        plan = await generate_diet_plan_ai(user, ans["answers"]) if False
else generate_diet_plan_rulebased(user, ans["answers"])  # toggle AI
    except Exception:
        plan = generate_diet_plan_rulebased(user, ans["answers"])
    doc = {"user_id": req.user_id, "plan": plan, "generated_at":
datetime.utcnow(), "summary": {}, "metadata": {"method": "rulebased"}}
    res = await diet_plans.insert_one(doc)
    doc["_id"] = str(res.inserted_id)
    return doc

@router.get("/{user_id}")
async def list_plans(user_id: str, token = Depends(verify_token)):
    cursor = diet_plans.find({"user_id": user_id}).sort("generated_at", -1)
    out = []
    async for d in cursor:
        d["_id"] = str(d["_id"])
        out.append(d)
    return out
```

backend/app/api/chatbot.py

```
from fastapi import APIRouter, Depends
from pydantic import BaseModel
from app.core.security import verify_token
from app.core.ai_client import chat_reply

router = APIRouter(prefix="/chat", tags=["chat"])
```

```
class ChatReq(BaseModel):
    message: str

@router.post("/message")
async def message(payload: ChatReq, token = Depends(verify_token)):
    # Very simple proxy to AI; in this starter it's a stubbed reply
    reply = await chat_reply(payload.message)
    return {"reply": reply}
```

---

backend/app/api/progress.py

```
from fastapi import APIRouter, Depends
from pydantic import BaseModel
from datetime import datetime
from app.db.mongo import progress
from app.core.security import verify_token

router = APIRouter(prefix="/progress", tags=["progress"])

class ProgressModel(BaseModel):
    user_id: str
    date: str
    weight_kg: float
    adherence_percent: int
    notes: str = ""

@router.post("/")
async def save_progress(payload: ProgressModel, token =
Depends(verify_token)):
    doc = payload.dict()
    doc["created_at"] = datetime.utcnow()
    res = await progress.insert_one(doc)
    doc["_id"] = str(res.inserted_id)
    return doc

@router.get("/{user_id}")
async def get_progress(user_id: str, token = Depends(verify_token)):
    cursor = progress.find({"user_id": user_id}).sort("date", 1)
    out = []
    async for d in cursor:
        d["_id"] = str(d["_id"])
        out.append(d)
    return out
```

---

backend/app/core/ai_client.py

```
# Lightweight AI client placeholder. If you have OPENAI_API_KEY set in .env
you can implement actual calls.
import os
import asyncio

async def chat_reply(message: str) -> str:
    # placeholder: echo with a friendly message. Replace with real AI API
call.
    return f"Bot: I received your message — '{message}'. (This is a starter
stub.)"


def generate_diet_plan_rulebased(user: dict, answers: list) -> dict:
    # Basic rule-based one-day diet plan; extend this logic to be more
sophisticated or call AI.
    weight = user.get("weight_kg", 70)
    height = user.get("height_cm", 170)
    age = user.get("age", 30)
    gender = user.get("gender", "other")

    meals = [
        {"time": "08:00", "meal_type": "Breakfast", "items": [{"name":
"Oats", "amount_g": 50, "calories": 200, "rasa": "Madhura", "virya": "Usna",
"vipaka": "Madhura"}]},
        {"time": "12:30", "meal_type": "Lunch", "items": [{"name": "Brown
Rice", "amount_g": 150, "calories": 210, "rasa": "Madhura", "virya":
"Sheeta", "vipaka": "Madhura"}, {"name": "Dal", "amount_g": 150, "calories":
180, "rasa": "Katu", "virya": "Usna", "vipaka": "Katu"}]},
        {"time": "17:00", "meal_type": "Snack", "items": [{"name": "Fruit
Salad", "amount_g": 150, "calories": 100, "rasa": "Madhura", "virya":
"Sheeta", "vipaka": "Madhura"}]},
        {"time": "20:00", "meal_type": "Dinner", "items": [{"name":
"Vegetable Curry", "amount_g": 300, "calories": 350, "rasa": "Katu", "virya":
"Usna", "vipaka": "Katu"}]}
    ]
    summary = {"total_calories": sum(i["items"][0]["calories"] if
len(i["items"])==1 else sum(it["calories"] for it in i["items"]) for i in
meals), "macro": {"carbs_g": 200, "protein_g": 70, "fat_g": 70}}
    return {"meals": meals, "summary": summary}
```

---

backend/app/utils/pdf_helper.py

```
# Simple server-side PDF helper (optional). This file is a placeholder for
more advanced server-side PDF generation.
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
from io import BytesIO


def plan_to_pdf_bytes(plan):
    buffer = BytesIO()
    c = canvas.Canvas(buffer, pagesize=letter)
    c.drawString(100, 750, "Diet Plan")
    y = 720
    for meal in plan.get("meals", []):
        c.drawString(80, y, f"{meal['time']} - {meal['meal_type']}")
        y -= 20
        for item in meal['items']:
            c.drawString(110, y, f"- {item['name']} ({item['amount_g']}g)
{item['calories']} kcal")
            y -= 18
        y -= 8
    c.showPage()
    c.save()
    pdf = buffer.getvalue()
    buffer.close()
    return pdf
```

FRONTEND

This frontend is a minimal React app using Vite/CRA-like structure. It uses Tailwind for quick styles.

frontend/package.json

```
{
  "name": "diet-planner-frontend",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "react-router-dom": "6",
    "axios": "1.4.0",
    "i18next": "23.0.0",
    "react-i18next": "13.0.0",
    "chart.js": "4.4.0",
    "react-chartjs-2": "5.2.0",
    "jspdf": "2.5.1",
    "html2canvas": "1.4.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test"
  }
}
```

---

frontend/Dockerfile

```
FROM node:20
WORKDIR /app
COPY package.json /app/
RUN npm install
COPY . /app
CMD ["npm", "start"]
```

---

frontend/public/index.html

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Diet Planner</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

---

frontend/src/index.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
import './index.css'

createRoot(document.getElementById('root')).render(<App />)
```

---

frontend/src/index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

body { font-family: ui-sans-serif, system-ui, -apple-system, 'Segoe UI',
Roboto, 'Helvetica Neue'; }
```

---

frontend/tailwind.config.js

```
module.exports = {
  content: ["./src/**/*.{js,jsx,ts,tsx}", "./public/index.html"],
  theme: { extend: {} },
  plugins: [],
}
```

---

frontend/src/api/apiClient.js

```
import axios from 'axios'

const api = axios.create({ baseURL: process.env.REACT_APP_API_URL || 'http://
localhost:8000', })

api.interceptors.request.use((config) => {
  const token = localStorage.getItem('access_token')
  if (token) config.headers.Authorization = `Bearer ${token}`
  return config
})

export default api
```

---

frontend/src/i18n.js

```
import i18n from 'i18next'
import { initReactI18next } from 'react-i18next'

const resources = {
  en: { translation: { "login": "Login", "signup": "Sign Up" } }
}
```

```
i18n.use(initReactI18next).init({ resources, lng: 'en', fallbackLng: 'en',
interpolation: { escapeValue: false } })
export default i18n
```

---

frontend/src/App.jsx

```
import React from 'react'
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom'
import Login from './pages/Login'
import Signup from './pages/Signup'
import Dashboard from './pages/Dashboard'
import AnalysisForm from './pages/AnalysisForm'
import Profile from './pages/Profile'

export default function App(){
  return (
    <BrowserRouter>
      <div className="min-h-screen bg-gray-50">
        <Routes>
          <Route path="/" element={<Login/>} />
          <Route path="/signup" element={<Signup/>} />
          <Route path="/dashboard" element={<Dashboard/>} />
          <Route path="/analysis" element={<AnalysisForm/>} />
          <Route path="/profile" element={<Profile/>} />
        </Routes>
      </div>
    </BrowserRouter>
  )
}
```

---

frontend/src/pages/Login.jsx

```
import React, {useState} from 'react'
import api from '../api/apiClient'

export default function Login(){
  const [email,setEmail] = useState('')
  const [password,setPassword] = useState('')
  async function handle(e){
    e.preventDefault()
    try{
      const res = await api.post('/auth/login',{email,password})
      localStorage.setItem('access_token', res.data.access_token)
      window.location.href = '/dashboard'
    }catch(err){
      alert('Login failed')
    }
  }
  return (
    <div className="max-w-md mx-auto p-8">
      <h2 className="text-2xl mb-4">Login</h2>
      <form onSubmit={handle} className="space-y-3">
        <input className="w-full p-2 border rounded" placeholder="Email"
value={email} onChange={e=>setEmail(e.target.value)} />
        <input className="w-full p-2 border rounded" placeholder="Password"
type="password" value={password} onChange={e=>setPassword(e.target.value)} />
        <button className="w-full p-2 bg-blue-600 text-white rounded">Login</
button>
      </form>
      <div className="mt-4">Don't have an account? <a href="/signup"
className="text-blue-600">Sign up</a></div>
    </div>
  )
}
```

frontend/src/pages/Signup.jsx

```
import React, {useState} from 'react'
import api from '../api/apiClient'

export default function Signup(){
  const [email,setEmail] = useState('')
  const [password,setPassword] = useState('')
  const [name,setName] = useState('')
  async function handle(e){
    e.preventDefault()
    try{
      const res = await api.post('/auth/signup',{email,password,name})
      localStorage.setItem('access_token', res.data.access_token)
      window.location.href = '/dashboard'
    }catch(err){
      alert('Signup failed')
    }
  }
  return (
    <div className="max-w-md mx-auto p-8">
      <h2 className="text-2xl mb-4">Sign Up</h2>
      <form onSubmit={handle} className="space-y-3">
        <input className="w-full p-2 border rounded" placeholder="Name"
value={name} onChange={e=>setName(e.target.value)} />
        <input className="w-full p-2 border rounded" placeholder="Email"
value={email} onChange={e=>setEmail(e.target.value)} />
        <input className="w-full p-2 border rounded" placeholder="Password"
type="password" value={password} onChange={e=>setPassword(e.target.value)} />
        <button className="w-full p-2 bg-green-600 text-white
rounded">Create Account</button>
      </form>
    </div>
  )
}
```

frontend/src/pages/AnalysisForm.jsx

```
import React, {useState} from 'react'
import api from '../api/apiClient'

export default function AnalysisForm(){
  const initial = Array.from({length:10}).map(()=>(''))
  const [answers,setAnswers] = useState(initial)
  const userId = localStorage.getItem('user_id') || null

  async function submitAnswers(){
    try{
      const saved = await api.post('/answers',{user_id: userId, answers})
      await api.post('/diet/generate',{user_id: userId, answers_id:
saved._id})
      window.location.href = '/dashboard'
    }catch(err){
      alert('Failed to submit answers')
    }
  }

  return (
    <div className="max-w-2xl mx-auto p-6">
      <h2 className="text-xl mb-4">Analysis Questions</h2>
      {answers.map((a,i)=>(
        <div key={i} className="mb-2">
          <label className="block">Question {i+1}</label>
          <input className="w-full p-2 border rounded" value={a}
onChange={e=>{const c=[...answers]; c[i]=e.target.value; setAnswers(c)}}/>
        </div>
      ))}
      <button onClick={submitAnswers} className="mt-4 p-2 bg-blue-600 text-
white rounded">Submit</button>
    </div>
  )
}
```

frontend/src/pages/Dashboard.jsx

```
import React, {useEffect, useState} from 'react'
import api from '../api/apiClient'
import NavMenu from '../components/NavMenu'
import DietTable from '../components/DietTable'
import ChartsPanel from '../components/ChartsPanel'
import ChatBot from '../components/ChatBot'

export default function Dashboard(){
  const [plans,setPlans] = useState([])
  const userId = localStorage.getItem('user_id')

  useEffect(()=>{
    async function load(){
      if(!userId) return
      try{
        const res = await api.get(`/diet/${userId}`)
        setPlans(res.data)
      }catch(err){ console.error(err) }
    }
    load()
  },[])

  return (
    <div>
      <NavMenu />
      <div className="p-6 grid grid-cols-1 md:grid-cols-2 gap-6">
        <div>
          <h3 className="text-xl">Latest Diet Plan</h3>
          {plans[0] ? <DietTable plan={plans[0].plan} /> : <div>No plan yet</
div>}
        </div>
        <div>
          <h3 className="text-xl">Charts</h3>
          <ChartsPanel plan={plans[0] ? plans[0].plan : null} />
        </div>
      </div>
      <div className="p-6"><ChatBot /></div>
    </div>
  )
}
```

---

frontend/src/pages/Profile.jsx

```
import React, {useState, useEffect} from 'react'
import api from '../api/apiClient'
```

```
export default function Profile(){
  const [profile, setProfile] = useState({})
  useEffect(()=>{
    // load profile from backend or local storage
  },[])
  return (
    <div className="max-w-2xl mx-auto p-6">
      <h2 className="text-xl mb-4">Profile</h2>
      <div>Email: {profile.email}</div>
      <div>Name: {profile.name}</div>
    </div>
  )
}
```

---

frontend/src/components/NavMenu.jsx

```
import React from 'react'

export default function NavMenu(){
  function logout(){
    localStorage.removeItem('access_token')
    localStorage.removeItem('user_id')
    window.location.href = '/'
  }
  return (
    <div className="flex items-center justify-between p-4 bg-white shadow">
      <div className="text-xl font-bold">Diet Planner</div>
      <div className="space-x-3">
        <a href="/profile" className="text-sm">Profile</a>
        <a href="/analysis" className="text-sm">Analysis</a>
        <button onClick={logout} className="ml-4 p-2 border rounded">Logout</
button>
      </div>
    </div>
  )
}
```

---

frontend/src/components/DietTable.jsx

```
import React from 'react'
import { jsPDF } from 'jspdf'
import html2canvas from 'html2canvas'

export default function DietTable({plan}){
  const exportPDF = async ()=>{
    const el = document.getElementById('diet-plan-for-pdf')
    if(!el) return
    const canvas = await html2canvas(el)
    const img = canvas.toDataURL('image/png')
    const pdf = new jsPDF()
    pdf.addImage(img, 'PNG', 10, 10, 190, 0)
    pdf.save('diet-plan.pdf')
  }

  if(!plan) return <div>No plan</div>
  return (
    <div>
      <div id="diet-plan-for-pdf" className="bg-white p-4 rounded shadow">
        {plan.meals.map((m,idx)=>(
          <div key={idx} className="mb-3">
            <div className="font-semibold">{m.time} □ {m.meal_type}</div>
            <ul className="ml-3">
              {m.items.map((it,i)=>(<li key={i}>{it.name} □ {it.amount_g}g □
{it.calories} kcal □ {it.rasa}/{it.virya}/{it.vipaka}</li>))}
            </ul>
          </div>
        ))}
      </div>
      <button onClick={exportPDF} className="mt-3 p-2 bg-indigo-600 text-
white rounded">Download PDF</button>
    </div>
  )
}
```

frontend/src/components/ChartsPanel.jsx

```
import React from 'react'
import { Pie } from 'react-chartjs-2'
import { Chart, ArcElement, Tooltip, Legend } from 'chart.js'
Chart.register(ArcElement, Tooltip, Legend)

export default function ChartsPanel({plan}){
  if(!plan) return <div>No chart</div>
  const macro = plan.summary?.macro || {carbs_g:200, protein_g:70, fat_g:60}
  const data = { labels:['Carbs','Protein','Fat'], datasets:[{ data:
[macro.carbs_g, macro.protein_g, macro.fat_g] }] }
  return (
    <div className="bg-white p-4 rounded shadow"> <Pie data={data} /> </div>
  )
}
```

frontend/src/components/ChatBot.jsx

```
import React, {useState} from 'react'
import api from '../api/apiClient'

export default function ChatBot(){
  const [text,setText] = useState('')
  const [messages,setMessages] = useState([])
  async function send(){
    if(!text) return
    setMessages(prev=>[...prev, {from:'user', text}])
    try{
      const res = await api.post('/chat/message', {message: text})
      setMessages(prev=>[...prev, {from:'user', text}, {from:'bot', text:
res.data.reply}])
    }catch(err){
      setMessages(prev=>[...prev, {from:'bot', text: 'Chat error'}])
    }
    setText('')
  }
  return (
    <div className="bg-white p-4 rounded shadow max-w-lg">
      <div className="h-48 overflow-y-auto mb-2">
        {messages.map((m,i)=>(<div key={i} className={m.from==='bot' ? 'text-
left' : 'text-right'}>{m.text}</div>))}
      </div>
      <div className="flex gap-2">
        <input value={text} onChange={e=>setText(e.target.value)}
className="flex-1 p-2 border rounded" />
        <button onClick={send} className="p-2 bg-green-600 text-white
rounded">Send</button>
      </div>
    </div>
  )
}
```

---

docker-compose.yml

```
version: '3.8'
services:
  mongodb:
    image: mongo:6
    restart: always
    ports:
      - '27017:27017'
    volumes:
      - mongo_data:/data/db
  backend:
    build: ./backend
    env_file: ./backend/.env
    ports:
      - '8000:8000'
    depends_on:
      - mongodb
  frontend:
    build: ./frontend
    ports:
      - '3000:3000'
    environment:
      - REACT_APP_API_URL=http://localhost:8000

volumes:
  mongo_data:
```

README.md (usage & DB management)

```
# Diet Planner — Starter

## Quick start (local with Docker)

1. Copy this repo to your machine.
2. Create `backend/.env` from `.env.sample` and set `JWT_SECRET` to a long
random string.
3. (Optional) add `OPENAI_API_KEY` if you plan to use AI.
4. Run:

   docker-compose up --build

5. Backend will be at http://localhost:8000 and frontend at http://localhost:
3000

## Run locally without Docker (backend)

1. cd backend
2. python -m venv venv
3. source venv/bin/activate
4. pip install -r requirements.txt
5. Create `.env` from `.env.sample`
6. uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

## MongoDB management (basic)

- To open a mongo shell on the Docker container:

  docker exec -it <mongo_container_id> mongosh
```

```
- Example: create index for users

  use dietdb
  db.users.createIndex({email:1},{unique:true})

- To inspect data from host machine with `mongosh`:

  mongosh mongodb://localhost:27017/dietdb

## Notes on production

- Use HTTPS
- Store JWT refresh tokens in HttpOnly cookies
- Use environment variables and secret management
- For heavy AI tasks, move to an async worker (Celery/RQ) and notify via
websockets
```

Final Notes & Next steps

1. The starter uses a rule-based diet generator (in `ai_client.py`). Replace it with real AI calls (OpenAI, etc.) by implementing `generate_diet_plan_ai` and toggling the call in `diet.py`.
2. Add proper validation, error handling, and tests before production.
3. If you want, I can now generate a ZIP of all these files or output the full content of any single file again for copying.

End of starter package.