Project Codebase Export

Generated automatically by export_codebase.py

Root

codebase.md

```

```

export_codebase.py

```
# export_codebase.py
import os

PROJECT_ROOT = "."
OUTPUT_FILE = "codebase.md"  # use Markdown for better PDF formatting
INCLUDE_EXTS = (".py", ".html", ".css", ".js", ".json", ".yml", ".sql", ".md",
".txt")

SKIP_CONTAINS = [".git", "__pycache__", "venv", ".venv", "env", "node_modules"]
SKIP_FILES = [".env", ".env.local", ".env.production"]

def should_skip_path(path):
    for s in SKIP_CONTAINS:
        if s in path:
            return True
    return False

with open(OUTPUT_FILE, "w", encoding="utf-8", errors="replace") as out:
    out.write("# 📦 Project Codebase Export\n")
    out.write("Generated automatically by export_codebase.py\n\n")

    for root, dirs, files in os.walk(PROJECT_ROOT):
        if should_skip_path(root):
            continue

        rel_path = os.path.relpath(root, PROJECT_ROOT)
        if rel_path == ".":
            rel_path = "Root"

        out.write(f"\n\n## 📁 {rel_path}\n\n")

        for file in sorted(files):
            if file in SKIP_FILES:
                out.write(f"🚫 Skipped secret file: `{file}`\n")
                continue
```

```
            if not file.lower().endswith(INCLUDE_EXTS):
                continue

            filepath = os.path.join(root, file)
            try:
                out.write(f"\n### 📄 {file}\n\n")
                out.write("```" + filepath.split(".")[-1] + "\n")  # code block
start
                with open(filepath, "r", encoding="utf-8", errors="replace") as f:
                    out.write(f.read())
                out.write("\n```\n")  # code block end
            except Exception as e:
                out.write(f"⚠️ Could not read {filepath}: {e}\n")

print(f"☑️ Export complete → {OUTPUT_FILE}")
print("Next step: run pandoc to make the PDF.")
```

requirements.txt

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
sqlalchemy==1.4.50
python-multipart==0.0.6
PyJWT==2.8.0
cryptography==41.0.7
passlib[bcrypt]==4.0.1
pytest==7.4.3
httpx==0.25.2
```

backend

init.py

```

```

auth.py

```
import os
from datetime import datetime, timedelta
from jose import jwt
from jose.exceptions import JWTError  # Changed this line
from passlib.context import CryptContext
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
```

```
from sqlalchemy.orm import Session
from dotenv import load_dotenv
from . import models
from .database import get_db
from datetime import datetime, timedelta, timezone
load_dotenv()

# Configuration - Use environment variables for security
SECRET_KEY = os.getenv("SECRET_KEY", "your-secret-key-here-change-in-production")
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = int(os.getenv("ACCESS_TOKEN_EXPIRE_MINUTES", "30"))

# Password hashing
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")

# --- Password Handling ---

def truncate_password(password: str) -> str:
    """Ensure password does not exceed bcrypt's 72-byte limit"""
    password_bytes = password.encode("utf-8")
    if len(password_bytes) > 72:
        return password_bytes[:72].decode("utf-8", errors="ignore")
    return password

def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verify a plain password against its hash"""
    plain_password = truncate_password(plain_password)
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:
    """Generate hash from plain password"""
    password = truncate_password(password)
    return pwd_context.hash(password)

# --- JWT Token Handling ---

def create_access_token(data: dict, expires_delta: timedelta = None) -> str:
    """Create JWT access token"""
    to_encode = data.copy()
    expire = datetime.utcnow() + (expires_delta or
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt

# --- Database Queries ---

def get_user_by_username(db: Session, username: str):
    """Get user by username from database"""
    return db.query(models.User).filter(models.User.username == username).first()

def get_user_by_email(db: Session, email: str):
```

```
    """Get user by email from database"""
    return db.query(models.User).filter(models.User.email == email).first()

def authenticate_user(db: Session, username: str, password: str):
    """Authenticate user with username and password"""
    user = get_user_by_username(db, username)
    if not user:
        return None
    if not verify_password(password, user.hashed_password):
        return None
    return user

# --- FastAPI Dependencies ---

def get_current_user(
    token: str = Depends(oauth2_scheme),
    db: Session = Depends(get_db)
):
    """Get current authenticated user from JWT token"""
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )

    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception

    user = get_user_by_username(db, username)
    if user is None:
        raise credentials_exception
    return user

def get_current_active_user(
    current_user: models.User = Depends(get_current_user)
):
    """Get current active user"""
    return current_user



def create_access_token(data: dict, expires_delta: timedelta = None) -> str:
    to_encode = data.copy()
    expire = datetime.now(timezone.utc) + (expires_delta or
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES))
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
```

crud.py

```
from sqlalchemy.orm import Session
from sqlalchemy import or_, and_
from typing import List, Optional
from . import models, schemas, auth

# ----------------- User CRUD -----------------
def create_user(db: Session, user: schemas.UserCreate) -> models.User:
    """Create a new user"""
    # Check for existing email or username
    if get_user_by_email(db, user.email) or get_user_by_username(db,
user.username):
        raise ValueError("Email or username already registered")

    hashed_password = auth.get_password_hash(user.password)
    db_user = models.User(
        username=user.username,
        email=user.email,
        hashed_password=hashed_password
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user


def get_user(db: Session, user_id: int) -> Optional[models.User]:
    """Get user by ID"""
    return db.query(models.User).filter(models.User.id == user_id).first()

def get_user_by_email(db: Session, email: str) -> Optional[models.User]:
    """Get user by email"""
    return db.query(models.User).filter(models.User.email == email).first()

def get_user_by_username(db: Session, username: str) -> Optional[models.User]:
    """Get user by username"""
    return db.query(models.User).filter(models.User.username == username).first()

# ----------------- Recipe CRUD -----------------
def create_recipe(db: Session, recipe: schemas.RecipeCreate, user_id: int) ->
models.Recipe:
    """Create a new recipe"""
    db_recipe = models.Recipe(**recipe.model_dump(), owner_id=user_id)
    db.add(db_recipe)
    db.commit()
    db.refresh(db_recipe)
    return db_recipe

def get_recipe(db: Session, recipe_id: int) -> Optional[models.Recipe]:
```

```
    """Get recipe by ID"""
    return db.query(models.Recipe).filter(models.Recipe.id == recipe_id).first()

def get_recipes(db: Session, skip: int = 0, limit: int = 100) ->
List[models.Recipe]:
    """Get all recipes with pagination"""
    return db.query(models.Recipe).offset(skip).limit(limit).all()

def get_user_recipes(db: Session, user_id: int, skip: int = 0, limit: int = 100) -
> List[models.Recipe]:
    """Get recipes by user ID"""
    return (
        db.query(models.Recipe)
        .filter(models.Recipe.owner_id == user_id)
        .offset(skip)
        .limit(limit)
        .all()
    )

def update_recipe(db: Session, recipe_id: int, recipe_update:
schemas.RecipeUpdate, user_id: int) -> Optional[models.Recipe]:
    """Update a recipe"""
    db_recipe = db.query(models.Recipe).filter(
        and_(models.Recipe.id == recipe_id, models.Recipe.owner_id == user_id)
    ).first()

    if not db_recipe:
        return None

    update_data = recipe_update.model_dump(exclude_unset=True)
    for field, value in update_data.items():
        setattr(db_recipe, field, value)

    db.commit()
    db.refresh(db_recipe)
    return db_recipe

def delete_recipe(db: Session, recipe_id: int, user_id: int) -> bool:
    """Delete a recipe"""
    db_recipe = db.query(models.Recipe).filter(
        and_(models.Recipe.id == recipe_id, models.Recipe.owner_id == user_id)
    ).first()

    if not db_recipe:
        return False

    db.delete(db_recipe)
    db.commit()
    return True

def search_recipes(db: Session, query: str, skip: int = 0, limit: int = 100) ->
List[models.Recipe]:
    """Search recipes by title, description, or ingredients"""
    return (
```

```
        db.query(models.Recipe)
        .filter(
            or_(
                models.Recipe.title.ilike(f"%{query}%"),
                models.Recipe.description.ilike(f"%{query}%"),
                models.Recipe.ingredients.ilike(f"%{query}%")
            )
        )
        .offset(skip)
        .limit(limit)
        .all()
    )

# ----------------- Favorite CRUD -----------------
def add_favorite(db: Session, user_id: int, recipe_id: int) -> models.Favorite:
    """Add recipe to favorites"""
    # Check if already favorited
    existing = db.query(models.Favorite).filter(
        and_(models.Favorite.user_id == user_id, models.Favorite.recipe_id ==
recipe_id)
    ).first()

    if existing:
        return existing

    db_favorite = models.Favorite(user_id=user_id, recipe_id=recipe_id)
    db.add(db_favorite)
    db.commit()
    db.refresh(db_favorite)
    return db_favorite

def remove_favorite(db: Session, user_id: int, recipe_id: int) -> bool:
    """Remove recipe from favorites"""
    db_favorite = db.query(models.Favorite).filter(
        and_(models.Favorite.user_id == user_id, models.Favorite.recipe_id ==
recipe_id)
    ).first()

    if not db_favorite:
        return False

    db.delete(db_favorite)
    db.commit()
    return True

def get_user_favorites(db: Session, user_id: int, skip: int = 0, limit: int = 100)
-> List[models.Favorite]:
    """Get user's favorite recipes"""
    return (
        db.query(models.Favorite)
        .filter(models.Favorite.user_id == user_id)
        .offset(skip)
        .limit(limit)
```

```
            .all()
        )
```

database.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import declarative_base, sessionmaker
import os
from dotenv import load_dotenv

load_dotenv()

# Use environment variable for database URL
SQLALCHEMY_DATABASE_URL = os.getenv("DATABASE_URL", "sqlite:///./recipes.db")

# Configure engine based on database type
if SQLALCHEMY_DATABASE_URL.startswith("sqlite"):
    engine = create_engine(
        SQLALCHEMY_DATABASE_URL,
        connect_args={"check_same_thread": False}
    )
else:
    # For PostgreSQL, MySQL, etc.
    engine = create_engine(SQLALCHEMY_DATABASE_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

def get_db():
    """Database dependency for dependency injection"""
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

main.py

```
from fastapi import FastAPI, HTTPException, Depends, status
from fastapi.middleware.cors import CORSMiddleware
from fastapi.security import OAuth2PasswordRequestForm
from fastapi.staticfiles import StaticFiles
from fastapi.responses import FileResponse
from sqlalchemy.orm import Session
from typing import List
from datetime import timedelta
import os
import httpx
```

```
from . import models, schemas, crud, auth
from .database import engine, get_db

# Create database tables
models.Base.metadata.create_all(bind=engine)

# Initialize FastAPI app
app = FastAPI(
    title="Recipe Finder API",
    description="A recipe management system with user authentication and CRUD
operations",
    version="1.0.0"
)

# CORS middleware for frontend integration
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:8000", "http://127.0.0.1:8000"],  # Frontend
URLs
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Mount static files (for frontend)
if os.path.exists("frontend"):
    app.mount("/static", StaticFiles(directory="frontend"), name="static")

# ----------------- Authentication Routes -----------------
@app.post("/auth/register", response_model=schemas.User)
async def register(user: schemas.UserCreate, db: Session = Depends(get_db)):
    """Register a new user"""
    # Check if user already exists
    if crud.get_user_by_email(db, user.email):
        raise HTTPException(
            status_code=400,
            detail="Email already registered"
        )
    if crud.get_user_by_username(db, user.username):
        raise HTTPException(
            status_code=400,
            detail="Username already taken"
        )

    return crud.create_user(db=db, user=user)

@app.post("/auth/login", response_model=schemas.Token)
async def login(form_data: OAuth2PasswordRequestForm = Depends(), db: Session =
Depends(get_db)):
    """Login user and return access token"""
    user = auth.authenticate_user(db, form_data.username, form_data.password)
    if not user:
        raise HTTPException(
```

```
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=auth.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = auth.create_access_token(
        data={"sub": user.username}, expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}

@app.get("/auth/me", response_model=schemas.User)
async def read_current_user(current_user: models.User =
Depends(auth.get_current_active_user)):
    """Get current user info"""
    return current_user

# ----------------- Recipe Routes -----------------
@app.get("/recipes")
async def read_all_recipes(skip: int = 0, limit: int = 10, db: Session =
Depends(get_db)):
    """Get all recipes (local + external API)"""
    # Local DB recipes
    local_recipes = crud.get_recipes(db, skip=skip, limit=limit)

    # External API recipes (random 5 for variety)
    async with httpx.AsyncClient() as client:
        res = await
client.get("https://www.themealdb.com/api/json/v1/1/search.php", params={"s": ""})
        data = res.json()
        external_recipes = data.get("meals", [])[:5] if data else []

    return {
        "local_recipes": local_recipes,
        "external_recipes": external_recipes
    }


@app.get("/recipes/search", response_model=List[schemas.Recipe])
async def search_recipes(q: str, skip: int = 0, limit: int = 20, db: Session =
Depends(get_db)):
    """Search recipes by title, description, or ingredients"""
    return crud.search_recipes(db, query=q, skip=skip, limit=limit)

@app.get("/recipes/my", response_model=List[schemas.Recipe])
async def read_my_recipes(
    skip: int = 0,
    limit: int = 100,
    current_user: models.User = Depends(auth.get_current_active_user),
    db: Session = Depends(get_db)
):
    """Get current user's recipes"""
    return crud.get_user_recipes(db, user_id=current_user.id, skip=skip,
limit=limit)
```

```python
@app.get("/recipes/{recipe_id}", response_model=schemas.Recipe)
async def read_recipe(recipe_id: int, db: Session = Depends(get_db)):
    """Get recipe by ID"""
    recipe = crud.get_recipe(db, recipe_id)
    if recipe is None:
        raise HTTPException(status_code=404, detail="Recipe not found")
    return recipe

@app.get("/recipes/search/ingredient")
async def search_recipes_by_ingredient(ingredient: str):
    """Search recipes from TheMealDB by ingredient"""
    async with httpx.AsyncClient() as client:
        res = await client.get(
            "https://www.themealdb.com/api/json/v1/1/filter.php",
            params={"i": ingredient}
        )
        data = res.json()
        if not data or not data.get("meals"):
            return {"meals": []}

        # Get details of first 5 results
        recipes = []
        for meal in data["meals"][:5]:
            lookup = await client.get(
                "https://www.themealdb.com/api/json/v1/1/lookup.php",
                params={"i": meal["idMeal"]}
            )
            recipes.append(lookup.json()["meals"][0])

        return {"meals": recipes}


@app.post("/recipes", response_model=schemas.Recipe)
async def create_recipe(
    recipe: schemas.RecipeCreate,
    current_user: models.User = Depends(auth.get_current_active_user),
    db: Session = Depends(get_db)
):
    """Create a new recipe"""
    return crud.create_recipe(db=db, recipe=recipe, user_id=current_user.id)

@app.put("/recipes/{recipe_id}", response_model=schemas.Recipe)
async def update_recipe(
    recipe_id: int,
    recipe: schemas.RecipeUpdate,
    current_user: models.User = Depends(auth.get_current_active_user),
    db: Session = Depends(get_db)
):
    """Update a recipe"""
    updated_recipe = crud.update_recipe(db, recipe_id, recipe, current_user.id)
    if updated_recipe is None:
        raise HTTPException(status_code=404, detail="Recipe not found or you don't
have permission")
    return updated_recipe
```

```python
@app.delete("/recipes/{recipe_id}")
async def delete_recipe(
    recipe_id: int,
    current_user: models.User = Depends(auth.get_current_active_user),
    db: Session = Depends(get_db)
):
    """Delete a recipe"""
    success = crud.delete_recipe(db, recipe_id, current_user.id)
    if not success:
        raise HTTPException(status_code=404, detail="Recipe not found or you don't
have permission")
    return {"message": "Recipe deleted successfully"}

# ----------------- Favorites Routes -----------------
@app.get("/favorites", response_model=List[schemas.Favorite])
async def read_favorites(
    skip: int = 0,
    limit: int = 100,
    current_user: models.User = Depends(auth.get_current_active_user),
    db: Session = Depends(get_db)
):
    """Get current user's favorite recipes"""
    return crud.get_user_favorites(db, user_id=current_user.id, skip=skip,
limit=limit)

@app.post("/favorites/{recipe_id}")
async def add_to_favorites(
    recipe_id: int,
    current_user: models.User = Depends(auth.get_current_active_user),
    db: Session = Depends(get_db)
):
    """Add recipe to favorites"""
    # Check if recipe exists
    recipe = crud.get_recipe(db, recipe_id)
    if not recipe:
        raise HTTPException(status_code=404, detail="Recipe not found")

    favorite = crud.add_favorite(db, current_user.id, recipe_id)
    return {"message": "Recipe added to favorites", "favorite_id": favorite.id}

@app.delete("/favorites/{recipe_id}")
async def remove_from_favorites(
    recipe_id: int,
    current_user: models.User = Depends(auth.get_current_active_user),
    db: Session = Depends(get_db)
):
    """Remove recipe from favorites"""
    success = crud.remove_favorite(db, current_user.id, recipe_id)
    if not success:
        raise HTTPException(status_code=404, detail="Favorite not found")
    return {"message": "Recipe removed from favorites"}

# ---------------- External API Integration ----------------
```

```python
# ---------------- External API Integration ----------------
@app.get("/api/recipes/external", tags=["External"])
async def search_external_recipes(q: str, number: int = 10):
    """Search recipes from TheMealDB by ingredient"""
    async with httpx.AsyncClient() as client:
        try:
            # Filter by ingredient
            filter_res = await client.get(
                "https://www.themealdb.com/api/json/v1/1/filter.php",
                params={"i": q},
                timeout=10.0
            )
            filter_data = filter_res.json()

            if not filter_data or not filter_data.get("meals"):
                return {"results": [], "total": 0}

            # Get full recipe details
            results = []
            for meal in filter_data["meals"][:number]:
                meal_id = meal["idMeal"]
                lookup_res = await client.get(
                    "https://www.themealdb.com/api/json/v1/1/lookup.php",
                    params={"i": meal_id},
                    timeout=10.0
                )
                meal_detail = lookup_res.json().get("meals", [{}])[0]

                # Build ingredients list
                ingredients = []
                for i in range(1, 21):
                    ingredient = meal_detail.get(f'strIngredient{i}')
                    measure = meal_detail.get(f'strMeasure{i}')
                    if ingredient and ingredient.strip():
                        ingredients.append(f"{measure} {ingredient}".strip())

                results.append({
                    'id': f"ext_{meal_detail['idMeal']}",
                    'title': meal_detail.get('strMeal', 'Unknown'),
                    'description': f"{meal_detail.get('strCategory', '')} -
{meal_detail.get('strArea', '')}".strip(' - '),
                    'image_url': meal_detail.get('strMealThumb'),
                    'instructions': meal_detail.get('strInstructions', 'No
instructions'),
                    'ingredients': '\n'.join(ingredients),
                    'difficulty': 'medium',
                    'external': True
                })

            return {'results': results, 'total': len(results)}

        except Exception as e:
            print(f"External API error: {e}")
            return {'results': [], 'total': 0}
```

```
# ----------------- Root Route -----------------
@app.get("/")
async def read_root():
    """Root endpoint"""
    if os.path.exists("frontend/index.html"):
        return FileResponse("frontend/index.html")
    return {
        "message": "Recipe Finder API",
        "docs": "/docs",
        "version": "1.0.0"
    }
#----------------- External API Category Route -----------------
@app.get("/api/recipes/category", tags=["External"])
async def get_recipes_by_category(c: str = "Seafood"):
    """Get recipes by category from TheMealDB"""
    async with httpx.AsyncClient() as client:
        try:
            response = await client.get(
                f"https://www.themealdb.com/api/json/v1/1/filter.php?c={c}",
                timeout=10.0
            )
            data = response.json()

            if not data or not data.get("meals"):
                return {"results": [], "total": 0}

            results = []
            for meal in data["meals"][:10]:  # Limit to 10 recipes per category
                results.append({
                    'id': f"ext_{meal['idMeal']}",
                    'title': meal.get('strMeal', 'Unknown'),
                    'image_url': meal.get('strMealThumb'),
                    'description': c,
                    'difficulty': 'medium',
                    'external': True
                })

            return {'results': results, 'total': len(results)}
        except Exception as e:
            print(f"Category API error: {e}")
            return {'results': [], 'total': 0}
```

models.py

```
from sqlalchemy import Column, Integer, String, Text, ForeignKey, DateTime,
Boolean, Float
from sqlalchemy.orm import relationship
from sqlalchemy.sql import func
```

```
from .database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    username = Column(String(50), unique=True, index=True, nullable=False)
    email = Column(String(100), unique=True, index=True, nullable=False)
    hashed_password = Column(String(255), nullable=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    is_active = Column(Boolean, default=True)

    # Relationships
    recipes = relationship("Recipe", back_populates="owner", cascade="all, delete-
orphan")
    favorites = relationship("Favorite", back_populates="user", cascade="all,
delete-orphan")

class Recipe(Base):
    __tablename__ = "recipes"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String(200), index=True, nullable=False)
    description = Column(Text)
    ingredients = Column(Text, nullable=False)  # JSON string or comma-separated
    instructions = Column(Text, nullable=False)
    prep_time = Column(Integer)  # in minutes
    cook_time = Column(Integer)  # in minutes
    servings = Column(Integer, default=1)
    difficulty = Column(String(20), default="easy")  # easy, medium, hard
    image_url = Column(String(500))
    source_url = Column(String(500))
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    updated_at = Column(DateTime(timezone=True), onupdate=func.now())

    # Foreign keys
    owner_id = Column(Integer, ForeignKey("users.id"), nullable=False)

    # Relationships
    owner = relationship("User", back_populates="recipes")
    favorites = relationship("Favorite", back_populates="recipe", cascade="all,
delete-orphan")

class Favorite(Base):
    __tablename__ = "favorites"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    recipe_id = Column(Integer, ForeignKey("recipes.id"), nullable=False)
    created_at = Column(DateTime(timezone=True), server_default=func.now())

    # Relationships
    user = relationship("User", back_populates="favorites")
    recipe = relationship("Recipe", back_populates="favorites")
```

schemas.py

```
from pydantic import BaseModel, EmailStr, field_validator, ConfigDict
from typing import List, Optional
from datetime import datetime
import re

# ----------------- Users -----------------
class UserBase(BaseModel):
    username: str
    email: str

class UserCreate(UserBase):
    password: str

    @field_validator('username')
    def validate_username(cls, v):
        if len(v) < 3:
            raise ValueError('Username must be at least 3 characters long')
        if not re.match(r'^[a-zA-Z0-9_]+$', v):
            raise ValueError('Username can only contain letters, numbers, and
underscores')
        return v

    @field_validator('password')
    def validate_password(cls, v):
        if len(v) < 6:
            raise ValueError('Password must be at least 6 characters long')
        return v

class User(UserBase):
    id: int
    is_active: bool
    created_at: datetime


    model_config = ConfigDict(from_attributes=True)

class UserWithRecipes(User):
    recipes: List['Recipe'] = []

# ----------------- Recipes -----------------
class RecipeBase(BaseModel):
    title: str
    description: Optional[str] = None
    ingredients: str
    instructions: str
    prep_time: Optional[int] = None
    cook_time: Optional[int] = None
    servings: Optional[int] = 1
```

```
    difficulty: Optional[str] = "easy"
    image_url: Optional[str] = None
    source_url: Optional[str] = None

class RecipeCreate(RecipeBase):
    pass

class RecipeUpdate(BaseModel):
    title: Optional[str] = None
    description: Optional[str] = None
    ingredients: Optional[str] = None
    instructions: Optional[str] = None
    prep_time: Optional[int] = None
    cook_time: Optional[int] = None
    servings: Optional[int] = None
    difficulty: Optional[str] = None
    image_url: Optional[str] = None
    source_url: Optional[str] = None

class Recipe(RecipeBase):
    id: int
    owner_id: int
    created_at: datetime
    updated_at: Optional[datetime] = None


    model_config = ConfigDict(from_attributes=True)

class RecipeWithOwner(Recipe):
    owner: User

# ----------------- Favorites -----------------
class FavoriteCreate(BaseModel):
    recipe_id: int

class Favorite(BaseModel):
    id: int
    user_id: int
    recipe_id: int
    created_at: datetime
    recipe: Recipe


    model_config = ConfigDict(from_attributes=True)

# ----------------- Auth -----------------
class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    username: Optional[str] = None

# ----------------- Search -----------------
```

```
class RecipeSearch(BaseModel):
    query: str
    diet: Optional[str] = None
    cuisine: Optional[str] = None
    max_ready_time: Optional[int] = None

# Fix forward references
UserWithRecipes.model_rebuild()
```

backend\tests

__init__.py

```

```

conftest.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
import pytest
from fastapi.testclient import TestClient
from backend.main import app, get_db
from backend.database import Base

import uuid

# Use in-memory SQLite for full isolation
SQLALCHEMY_TEST_DATABASE_URL = "sqlite:///./test_test.db"
engine = create_engine(SQLALCHEMY_TEST_DATABASE_URL, connect_args=
{"check_same_thread": False})
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Override FastAPI dependency
def override_get_db():
    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()

app.dependency_overrides[get_db] = override_get_db


# Fixture: create fresh database before tests
@pytest.fixture(scope="session", autouse=True)
def create_test_db():
    Base.metadata.create_all(bind=engine)
    yield
    Base.metadata.drop_all(bind=engine)
```

```
@pytest.fixture(scope="function")
def client():
    # Create tables before each test
    Base.metadata.create_all(bind=engine)
    yield TestClient(app)
    # Drop tables after each test
    Base.metadata.drop_all(bind=engine)


@pytest.fixture
def test_user(client):
    unique_id = uuid.uuid4().hex[:6]
    user_data = {
        "username": f"testuser_{unique_id}",
        "email": f"test_{unique_id}@example.com",
        "password": "password123"
    }
    response = client.post("/auth/register", json=user_data)
    assert response.status_code == 200, f"User registration failed:
{response.json()}"

    # login to get token
    login_resp = client.post("/auth/login", data={"username":
user_data["username"], "password": user_data["password"]})
    assert login_resp.status_code == 200
    token = login_resp.json()["access_token"]

    return {"user": user_data, "token": token, "headers": {"Authorization":
f"Bearer {token}"}}

@pytest.fixture(autouse=True)
def clear_db():
    yield
    from backend.models import User, Recipe, Favorite  # adjust imports
    db = TestingSessionLocal()
    db.query(Favorite).delete()
    db.query(Recipe).delete()
    db.query(User).delete()
    db.commit()
    db.close()

def test_create_recipe(client, test_user):
    headers = test_user["headers"]
    response = client.post(
        "/recipes",
        json={"title": "My Recipe", "ingredients": "Eggs", "instructions": "Cook
it"},
        headers=headers
    )
    assert response.status_code == 200
```

test_auth.py

```
import os
import sys
import pytest
from fastapi.testclient import TestClient
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
import jwt
from datetime import timedelta
from backend.database import Base, get_db
from backend.main import app
from backend import models, crud, schemas, auth



# Make sure the backend package is importable
backend_path = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
if backend_path not in sys.path:
    sys.path.insert(0, backend_path)







# Add backend directory to path
backend_path = os.path.abspath(os.path.join(os.path.dirname(__file__), '..'))
if backend_path not in sys.path:
    sys.path.insert(0, backend_path)


# Test database setup
SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread":
False})
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

@pytest.fixture
def db():
    """Create a fresh database for each test"""
    Base.metadata.create_all(bind=engine)
    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()
        Base.metadata.drop_all(bind=engine)

class TestPasswordHandling:
    """Test password hashing and verification"""
```

```
    def test_password_hash_and_verify(self):
        """Test that password can be hashed and verified"""
        password = "mysecretpassword123"
        hashed = auth.get_password_hash(password)

        assert hashed != password
        assert auth.verify_password(password, hashed)

    def test_password_hash_different_each_time(self):
        """Test that same password produces different hashes"""
        password = "samepassword"
        hash1 = auth.get_password_hash(password)
        hash2 = auth.get_password_hash(password)

        assert hash1 != hash2
        assert auth.verify_password(password, hash1)
        assert auth.verify_password(password, hash2)

    def test_wrong_password_fails(self):
        """Test that wrong password doesn't verify"""
        password = "correctpassword"
        wrong_password = "wrongpassword"
        hashed = auth.get_password_hash(password)

        assert not auth.verify_password(wrong_password, hashed)

    def test_truncate_long_password(self):
        """Test that passwords longer than 72 bytes are truncated"""
        long_password = "a" * 100
        truncated = auth.truncate_password(long_password)

        assert len(truncated.encode('utf-8')) <= 72

class TestJWTTokens:
    """Test JWT token creation and validation"""

    def test_create_access_token(self):
        """Test access token creation"""
        data = {"sub": "testuser"}
        token = auth.create_access_token(data)

        assert token is not None
        assert isinstance(token, str)

    def test_token_contains_correct_data(self):
        """Test that token contains the encoded data"""
        username = "testuser123"
        data = {"sub": username}
        token = auth.create_access_token(data)

        payload = jwt.decode(token, auth.SECRET_KEY, algorithms=[auth.ALGORITHM])
        assert payload["sub"] == username
        assert "exp" in payload
```

```
    def test_token_with_custom_expiration(self):
        """Test token creation with custom expiration"""
        data = {"sub": "testuser"}
        expires_delta = timedelta(minutes=15)
        token = auth.create_access_token(data, expires_delta)

        payload = jwt.decode(token, auth.SECRET_KEY, algorithms=[auth.ALGORITHM])
        assert "exp" in payload

class TestUserQueries:
    """Test database user queries"""

    def test_get_user_by_username(self, db):
        """Test retrieving user by username"""
        # Create test user
        user = models.User(
            username="testuser",
            email="test@example.com",
            hashed_password=auth.get_password_hash("password123")
        )
        db.add(user)
        db.commit()

        # Query user
        found_user = auth.get_user_by_username(db, "testuser")
        assert found_user is not None
        assert found_user.username == "testuser"
        assert found_user.email == "test@example.com"

    def test_get_user_by_email(self, db):
        """Test retrieving user by email"""
        user = models.User(
            username="testuser",
            email="test@example.com",
            hashed_password=auth.get_password_hash("password123")
        )
        db.add(user)
        db.commit()

        found_user = auth.get_user_by_email(db, "test@example.com")
        assert found_user is not None
        assert found_user.email == "test@example.com"

    def test_get_nonexistent_user_returns_none(self, db):
        """Test that querying nonexistent user returns None"""
        user = auth.get_user_by_username(db, "nonexistent")
        assert user is None

class TestUserAuthentication:
    """Test user authentication"""

    def test_authenticate_user_success(self, db):
        """Test successful authentication"""
        password = "mypassword123"
```

```
        user = models.User(
            username="testuser",
            email="test@example.com",
            hashed_password=auth.get_password_hash(password)
        )
        db.add(user)
        db.commit()

        authenticated = auth.authenticate_user(db, "testuser", password)
        assert authenticated is not None
        assert authenticated.username == "testuser"

    def test_authenticate_user_wrong_password(self, db):
        """Test authentication with wrong password"""
        user = models.User(
            username="testuser",
            email="test@example.com",
            hashed_password=auth.get_password_hash("correctpassword")
        )
        db.add(user)
        db.commit()

        authenticated = auth.authenticate_user(db, "testuser", "wrongpassword")
        assert authenticated is None

    def test_authenticate_nonexistent_user(self, db):
        """Test authentication with nonexistent username"""
        authenticated = auth.authenticate_user(db, "nonexistent", "password")
        assert authenticated is None

if __name__ == "__main__":
    pytest.main([__file__, "-v"])
```

test_main.py

```
import pytest
from fastapi.testclient import TestClient
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from ..database import Base, get_db
from ..main import app
from .. import models, crud, schemas


# Test database setup
SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread":
False})
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def override_get_db():
```

```
    """Override database dependency for testing"""
    try:
        db = TestingSessionLocal()
        yield db
    finally:
        db.close()

app.dependency_overrides[get_db] = override_get_db

@pytest.fixture
def client():
    """Create test client"""
    Base.metadata.create_all(bind=engine)
    yield TestClient(app)
    Base.metadata.drop_all(bind=engine)

@pytest.fixture
def test_user(client):
    """Create and return test user with token"""
    user_data = {
        "username": "testuser",
        "email": "test@example.com",
        "password": "password123"
    }
    response = client.post("/auth/register", json=user_data)
    assert response.status_code == 200

    # Login to get token
    login_data = {
        "username": "testuser",
        "password": "password123"
    }
    response = client.post("/auth/login", data=login_data)
    token = response.json()["access_token"]

    return {"token": token, "user": user_data}

class TestAuthenticationRoutes:
    """Test authentication endpoints"""

    def test_register_user(self, client):
        """Test user registration"""
        user_data = {
            "username": "newuser",
            "email": "newuser@example.com",
            "password": "newpassword123"
        }
        response = client.post("/auth/register", json=user_data)

        assert response.status_code == 200
        data = response.json()
        assert data["username"] == "newuser"
        assert data["email"] == "newuser@example.com"
        assert "id" in data
```

```
    def test_register_duplicate_email(self, client, test_user):
        """Test registering with duplicate email fails"""
        user_data = {
            "username": "differentuser",
            "email": "test@example.com",
            "password": "password123"
        }
        response = client.post("/auth/register", json=user_data)

        assert response.status_code == 400
        assert "Email already registered" in response.json()["detail"]

    def test_register_duplicate_username(self, client, test_user):
        """Test registering with duplicate username fails"""
        user_data = {
            "username": "testuser",
            "email": "different@example.com",
            "password": "password123"
        }
        response = client.post("/auth/register", json=user_data)

        assert response.status_code == 400
        assert "Username already taken" in response.json()["detail"]

    def test_login_success(self, client, test_user):
        """Test successful login"""
        login_data = {
            "username": "testuser",
            "password": "password123"
        }
        response = client.post("/auth/login", data=login_data)

        assert response.status_code == 200
        data = response.json()
        assert "access_token" in data
        assert data["token_type"] == "bearer"

    def test_login_wrong_password(self, client, test_user):
        """Test login with wrong password"""
        login_data = {
            "username": "testuser",
            "password": "wrongpassword"
        }
        response = client.post("/auth/login", data=login_data)

        assert response.status_code == 401

    def test_login_nonexistent_user(self, client):
        """Test login with nonexistent username"""
        login_data = {
            "username": "nonexistent",
            "password": "password123"
        }
```

```
        response = client.post("/auth/login", data=login_data)

        assert response.status_code == 401

    def test_get_current_user(self, client, test_user):
        """Test getting current user info"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}
        response = client.get("/auth/me", headers=headers)

        assert response.status_code == 200
        data = response.json()
        assert data["username"] == "testuser"
        assert data["email"] == "test@example.com"

    def test_get_current_user_no_token(self, client):
        """Test getting current user without token fails"""
        response = client.get("/auth/me")
        assert response.status_code == 401

class TestRecipeRoutes:
    """Test recipe endpoints"""

    def test_create_recipe(self, client, test_user):
        """Test creating a recipe"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}
        recipe_data = {
            "title": "Test Recipe",
            "description": "A delicious recipe",
            "ingredients": "flour, eggs, milk",
            "instructions": "Mix and bake",
            "prep_time": 15,
            "cook_time": 30,
            "servings": 4,
            "difficulty": "easy"
        }
        response = client.post("/recipes", json=recipe_data, headers=headers)

        assert response.status_code == 200
        data = response.json()
        assert data["title"] == "Test Recipe"
        assert "id" in data

    def test_create_recipe_unauthorized(self, client):
        """Test creating recipe without authentication"""
        recipe_data = {
            "title": "Test Recipe",
            "ingredients": "ingredients",
            "instructions": "instructions"
        }
        response = client.post("/recipes", json=recipe_data)

        assert response.status_code == 401

    def test_get_recipe(self, client, test_user):
```

```
        """Test getting a recipe by ID"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create recipe
        recipe_data = {
            "title": "Test Recipe",
            "ingredients": "ingredients",
            "instructions": "instructions"
        }
        create_response = client.post("/recipes", json=recipe_data,
headers=headers)
        recipe_id = create_response.json()["id"]

        # Get recipe
        response = client.get(f"/recipes/{recipe_id}")
        assert response.status_code == 200
        data = response.json()
        assert data["id"] == recipe_id

    def test_get_nonexistent_recipe(self, client):
        """Test getting nonexistent recipe returns 404"""
        response = client.get("/recipes/99999")
        assert response.status_code == 404

    def test_get_all_recipes(self, client, test_user):
        """Test getting all recipes"""
        response = client.get("/recipes")

        assert response.status_code == 200
        data = response.json()
        assert "local_recipes" in data
        assert "external_recipes" in data

    def test_get_my_recipes(self, client, test_user):
        """Test getting current user's recipes"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create recipes
        for i in range(3):
            recipe_data = {
                "title": f"My Recipe {i}",
                "ingredients": "ingredients",
                "instructions": "instructions"
            }
            client.post("/recipes", json=recipe_data, headers=headers)

        # Get my recipes
        response = client.get("/recipes/my", headers=headers)
        assert response.status_code == 200
        data = response.json()
        assert len(data) == 3

    def test_search_recipes(self, client, test_user):
        """Test searching recipes"""
```

```
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create test recipes
        recipes = [
            {"title": "Chocolate Cake", "ingredients": "chocolate",
"instructions": "bake"},
            {"title": "Vanilla Cookies", "ingredients": "vanilla", "instructions":
"bake"}
        ]
        for recipe_data in recipes:
            client.post("/recipes", json=recipe_data, headers=headers)

        # Search
        response = client.get("/recipes/search", params={"q": "chocolate"})
        assert response.status_code == 200
        data = response.json()
        assert len(data) >= 1
        assert any("chocolate" in r["title"].lower() for r in data)

    def test_update_recipe(self, client, test_user):
        """Test updating a recipe"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create recipe
        recipe_data = {
            "title": "Original Title",
            "ingredients": "ingredients",
            "instructions": "instructions"
        }
        create_response = client.post("/recipes", json=recipe_data,
headers=headers)
        recipe_id = create_response.json()["id"]

        # Update recipe
        update_data = {"title": "Updated Title"}
        response = client.put(f"/recipes/{recipe_id}", json=update_data,
headers=headers)

        assert response.status_code == 200
        data = response.json()
        assert data["title"] == "Updated Title"

    def test_update_recipe_unauthorized(self, client, test_user):
        """Test updating recipe without proper authorization"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create recipe
        recipe_data = {
            "title": "Recipe",
            "ingredients": "ingredients",
            "instructions": "instructions"
        }
        create_response = client.post("/recipes", json=recipe_data,
headers=headers)
```

```
        recipe_id = create_response.json()["id"]

        # Try to update without token
        update_data = {"title": "Hacked"}
        response = client.put(f"/recipes/{recipe_id}", json=update_data)

        assert response.status_code == 401

    def test_delete_recipe(self, client, test_user):
        """Test deleting a recipe"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create recipe
        recipe_data = {
            "title": "To Delete",
            "ingredients": "ingredients",
            "instructions": "instructions"
        }
        create_response = client.post("/recipes", json=recipe_data,
headers=headers)
        recipe_id = create_response.json()["id"]

        # Delete recipe
        response = client.delete(f"/recipes/{recipe_id}", headers=headers)
        assert response.status_code == 200

        # Verify deleted
        get_response = client.get(f"/recipes/{recipe_id}")
        assert get_response.status_code == 404

    def test_delete_recipe_unauthorized(self, client, test_user):
        """Test deleting recipe without authorization"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create recipe
        recipe_data = {
            "title": "Recipe",
            "ingredients": "ingredients",
            "instructions": "instructions"
        }
        create_response = client.post("/recipes", json=recipe_data,
headers=headers)
        recipe_id = create_response.json()["id"]

        # Try to delete without token
        response = client.delete(f"/recipes/{recipe_id}")
        assert response.status_code == 401

class TestFavoriteRoutes:
    """Test favorite endpoints"""

    def test_add_favorite(self, client, test_user):
        """Test adding recipe to favorites"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}
```

```
        # Create recipe
        recipe_data = {
            "title": "Favorite Recipe",
            "ingredients": "ingredients",
            "instructions": "instructions"
        }
        create_response = client.post("/recipes", json=recipe_data,
headers=headers)
        recipe_id = create_response.json()["id"]

        # Add to favorites
        response = client.post(f"/favorites/{recipe_id}", headers=headers)
        assert response.status_code == 200
        assert "message" in response.json()

    def test_add_nonexistent_recipe_to_favorites(self, client, test_user):
        """Test adding nonexistent recipe to favorites"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}
        response = client.post("/favorites/99999", headers=headers)
        assert response.status_code == 404

    def test_get_favorites(self, client, test_user):
        """Test getting user's favorites"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create and favorite recipes
        for i in range(3):
            recipe_data = {
                "title": f"Recipe {i}",
                "ingredients": "ingredients",
                "instructions": "instructions"
            }
            create_response = client.post("/recipes", json=recipe_data,
headers=headers)
            recipe_id = create_response.json()["id"]
            client.post(f"/favorites/{recipe_id}", headers=headers)

        # Get favorites
        response = client.get("/favorites", headers=headers)
        assert response.status_code == 200
        data = response.json()
        assert len(data) == 3

    def test_remove_favorite(self, client, test_user):
        """Test removing recipe from favorites"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}

        # Create and favorite recipe
        recipe_data = {
            "title": "Recipe",
            "ingredients": "ingredients",
            "instructions": "instructions"
        }
```

```
        create_response = client.post("/recipes", json=recipe_data,
headers=headers)
        recipe_id = create_response.json()["id"]
        client.post(f"/favorites/{recipe_id}", headers=headers)

        # Remove from favorites
        response = client.delete(f"/favorites/{recipe_id}", headers=headers)
        assert response.status_code == 200

        # Verify removed
        favorites_response = client.get("/favorites", headers=headers)
        assert len(favorites_response.json()) == 0

    def test_remove_nonexistent_favorite(self, client, test_user):
        """Test removing nonexistent favorite"""
        headers = {"Authorization": f"Bearer {test_user['token']}"}
        response = client.delete("/favorites/99999", headers=headers)
        assert response.status_code == 404

class TestRootRoute:
    """Test root endpoint"""

    def test_root_endpoint(self, client):
        """Test root endpoint returns expected response"""
        response = client.get("/")
        assert response.status_code == 200

if __name__ == "__main__":
    pytest.main([__file__, "-v"])
```

test_recipes.py

```
import pytest
from fastapi.testclient import TestClient
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from ..database import Base, get_db
from ..main import app
from .. import models, crud, schemas

# Create test database
SQLALCHEMY_DATABASE_URL = "sqlite:///./test_recipes.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread":
False})
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base.metadata.create_all(bind=engine)

def override_get_db():
    try:
```

```
        db = TestingSessionLocal()
        yield db
    finally:
        db.close()

app.dependency_overrides[get_db] = override_get_db
client = TestClient(app)

# Helper function to create and login a user, return auth header
def create_auth_user(username="testuser", email="test@example.com",
password="testpass123"):
    client.post(
        "/auth/register",
        json={"username": username, "email": email, "password": password}
    )
    response = client.post("/auth/login", data={"username": username, "password":
password})
    token = response.json()["access_token"]
    return {"Authorization": f"Bearer {token}"}

class TestRecipeCreation:
    """Test recipe creation functionality"""

    def test_create_recipe_success(self):
        """Test successful recipe creation"""
        headers = create_auth_user("creator1", "creator1@test.com")
        response = client.post(
            "/recipes",
            json={
                "title": "Test Recipe",
                "description": "A delicious test recipe",
                "ingredients": "flour, sugar, eggs",
                "instructions": "Mix and bake",
                "prep_time": 15,
                "cook_time": 30,
                "servings": 4,
                "difficulty": "easy"
            },
            headers=headers
        )
        assert response.status_code == 200
        data = response.json()
        assert data["title"] == "Test Recipe"
        assert data["prep_time"] == 15
        assert "id" in data

    def test_create_recipe_minimal(self):
        """Test creating recipe with minimal fields"""
        headers = create_auth_user("creator2", "creator2@test.com")
        response = client.post(
            "/recipes",
            json={
                "title": "Simple Recipe",
                "ingredients": "ingredient1, ingredient2",
```

```
                "instructions": "Mix and cook"
            },
            headers=headers
        )
        assert response.status_code == 200
        assert response.json()["title"] == "Simple Recipe"

    def test_create_recipe_unauthorized(self):
        """Test creating recipe without authentication"""
        response = client.post(
            "/recipes",
            json={
                "title": "Unauthorized Recipe",
                "ingredients": "test",
                "instructions": "test"
            }
        )
        assert response.status_code == 401

class TestRecipeRetrieval:
    """Test recipe retrieval"""

    def test_get_all_recipes(self):
        """Test getting all recipes"""
        response = client.get("/recipes")
        assert response.status_code == 200
        data = response.json()
        assert "local_recipes" in data or isinstance(data, list)

    def test_get_my_recipes(self):
        """Test getting current user's recipes"""
        headers = create_auth_user("owner1", "owner1@test.com")

        # Create recipe
        client.post(
            "/recipes",
            json={"title": "My Recipe", "ingredients": "test", "instructions":
"test"},
            headers=headers
        )

        # Get my recipes
        response = client.get("/recipes/my", headers=headers)
        assert response.status_code == 200
        data = response.json()
        assert len(data) >= 1

    def test_get_recipe_by_id(self):
        """Test getting specific recipe"""
        headers = create_auth_user("getter1", "getter1@test.com")

        # Create recipe
        create_resp = client.post(
            "/recipes",
```

```
            json={"title": "Get Me", "ingredients": "test", "instructions":
"test"},
            headers=headers
        )
        recipe_id = create_resp.json()["id"]

        # Get recipe
        response = client.get(f"/recipes/{recipe_id}")
        assert response.status_code == 200
        assert response.json()["id"] == recipe_id

    def test_get_nonexistent_recipe(self):
        """Test getting non-existent recipe"""
        response = client.get("/recipes/99999")
        assert response.status_code == 404

class TestRecipeSearch:
    """Test recipe search"""

    def test_search_by_title(self):
        """Test searching recipes by title"""
        headers = create_auth_user("searcher1", "searcher1@test.com")

        # Create test recipe
        client.post(
            "/recipes",
            json={
                "title": "Chocolate Cake",
                "ingredients": "chocolate, flour",
                "instructions": "bake"
            },
            headers=headers
        )

        # Search
        response = client.get("/recipes/search?q=chocolate")
        assert response.status_code == 200
        results = response.json()
        assert len(results) >= 1

    def test_search_by_ingredients(self):
        """Test searching by ingredients"""
        headers = create_auth_user("searcher2", "searcher2@test.com")

        client.post(
            "/recipes",
            json={
                "title": "Chicken Dish",
                "ingredients": "chicken, garlic, onions",
                "instructions": "cook"
            },
            headers=headers
        )
```

```
        response = client.get("/recipes/search?q=chicken")
        assert response.status_code == 200
        assert len(response.json()) >= 1

class TestRecipeUpdate:
    """Test recipe updates"""

    def test_update_recipe(self):
        """Test updating a recipe"""
        headers = create_auth_user("updater1", "updater1@test.com")

        # Create recipe
        create_resp = client.post(
            "/recipes",
            json={
                "title": "Original",
                "ingredients": "original",
                "instructions": "original"
            },
            headers=headers
        )
        recipe_id = create_resp.json()["id"]

        # Update
        response = client.put(
            f"/recipes/{recipe_id}",
            json={"title": "Updated Title"},
            headers=headers
        )
        assert response.status_code == 200
        assert response.json()["title"] == "Updated Title"

    def test_update_unauthorized(self):
        """Test updating without auth"""
        headers = create_auth_user("updater2", "updater2@test.com")

        create_resp = client.post(
            "/recipes",
            json={"title": "Recipe", "ingredients": "test", "instructions":
"test"},
            headers=headers
        )
        recipe_id = create_resp.json()["id"]

        # Try update without auth
        response = client.put(
            f"/recipes/{recipe_id}",
            json={"title": "Hacked"}
        )
        assert response.status_code == 401

class TestRecipeDeletion:
    """Test recipe deletion"""
```

```
    def test_delete_recipe(self):
        """Test deleting a recipe"""
        headers = create_auth_user("deleter1", "deleter1@test.com")

        # Create recipe
        create_resp = client.post(
            "/recipes",
            json={"title": "Delete Me", "ingredients": "test", "instructions":
"test"},
            headers=headers
        )
        recipe_id = create_resp.json()["id"]

        # Delete
        response = client.delete(f"/recipes/{recipe_id}", headers=headers)
        assert response.status_code == 200

        # Verify deleted
        get_resp = client.get(f"/recipes/{recipe_id}")
        assert get_resp.status_code == 404

    def test_delete_unauthorized(self):
        """Test deleting without auth"""
        headers = create_auth_user("deleter2", "deleter2@test.com")

        create_resp = client.post(
            "/recipes",
            json={"title": "Recipe", "ingredients": "test", "instructions":
"test"},
            headers=headers
        )
        recipe_id = create_resp.json()["id"]

        # Try delete without auth
        response = client.delete(f"/recipes/{recipe_id}")
        assert response.status_code == 401

class TestFavorites:
    """Test favorite functionality"""

    def test_add_favorite(self):
        """Test adding recipe to favorites"""
        headers = create_auth_user("fav1", "fav1@test.com")

        # Create recipe
        create_resp = client.post(
            "/recipes",
            json={"title": "Favorite Recipe", "ingredients": "test",
"instructions": "test"},
            headers=headers
        )
        recipe_id = create_resp.json()["id"]

        # Add to favorites
```

```
        response = client.post(f"/favorites/{recipe_id}", headers=headers)
        assert response.status_code == 200

    def test_remove_favorite(self):
        """Test removing from favorites"""
        headers = create_auth_user("fav2", "fav2@test.com")

        # Create and favorite
        create_resp = client.post(
            "/recipes",
            json={"title": "Recipe", "ingredients": "test", "instructions":
"test"},
            headers=headers
        )
        recipe_id = create_resp.json()["id"]
        client.post(f"/favorites/{recipe_id}", headers=headers)

        # Remove favorite
        response = client.delete(f"/favorites/{recipe_id}", headers=headers)
        assert response.status_code == 200

    def test_get_favorites(self):
        """Test getting user favorites"""
        headers = create_auth_user("fav3", "fav3@test.com")

        # Create and favorite recipe
        create_resp = client.post(
            "/recipes",
            json={"title": "Fav Recipe", "ingredients": "test", "instructions":
"test"},
            headers=headers
        )
        recipe_id = create_resp.json()["id"]
        client.post(f"/favorites/{recipe_id}", headers=headers)

        # Get favorites
        response = client.get("/favorites", headers=headers)
        assert response.status_code == 200
        assert len(response.json()) >= 1
```

test_users.py

```
import pytest
from fastapi.testclient import TestClient
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from ..database import Base, get_db
from ..main import app
from .. import models, crud, schemas, auth

# Create test database
```

```
SQLALCHEMY_DATABASE_URL = "sqlite:///./test_users.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread":
False})
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base.metadata.create_all(bind=engine)

def override_get_db():
    try:
        db = TestingSessionLocal()
        yield db
    finally:
        db.close()

app.dependency_overrides[get_db] = override_get_db
client = TestClient(app)

class TestUserRegistration:
    """Test user registration"""

    def test_register_valid_user(self):
        """Test registering a valid user"""
        response = client.post(
            "/auth/register",
            json={
                "username": "newuser1",
                "email": "newuser1@example.com",
                "password": "password123"
            }
        )
        assert response.status_code == 200
        data = response.json()
        assert data["username"] == "newuser1"
        assert data["email"] == "newuser1@example.com"
        assert "id" in data
        assert "hashed_password" not in data
        assert data["is_active"] is True

    def test_register_duplicate_email(self):
        """Test duplicate email fails"""
        client.post(
            "/auth/register",
            json={
                "username": "user1",
                "email": "duplicate@example.com",
                "password": "password123"
            }
        )
        response = client.post(
            "/auth/register",
            json={
                "username": "user2",
                "email": "duplicate@example.com",
                "password": "password456"
```

```
            }
        )
        assert response.status_code == 400
        assert "email" in response.json()["detail"].lower()

    def test_register_duplicate_username(self):
        """Test duplicate username fails"""
        client.post(
            "/auth/register",
            json={
                "username": "sameuser",
                "email": "user1@example.com",
                "password": "password123"
            }
        )
        response = client.post(
            "/auth/register",
            json={
                "username": "sameuser",
                "email": "user2@example.com",
                "password": "password456"
            }
        )
        assert response.status_code == 400
        assert "username" in response.json()["detail"].lower()

    def test_register_short_username(self):
        """Test short username validation"""
        response = client.post(
            "/auth/register",
            json={
                "username": "ab",
                "email": "test@example.com",
                "password": "password123"
            }
        )
        assert response.status_code == 422

    def test_register_short_password(self):
        """Test short password validation"""
        response = client.post(
            "/auth/register",
            json={
                "username": "testuser",
                "email": "test@example.com",
                "password": "12345"
            }
        )
        assert response.status_code == 422

    def test_register_invalid_username_chars(self):
        """Test invalid username characters"""
        response = client.post(
            "/auth/register",
```

```
            json={
                "username": "user@name!",
                "email": "test@example.com",
                "password": "password123"
            }
        )
        assert response.status_code == 422

class TestUserLogin:
    """Test user login"""

    def test_login_success(self):
        """Test successful login"""
        client.post(
            "/auth/register",
            json={
                "username": "loginuser",
                "email": "login@example.com",
                "password": "loginpass123"
            }
        )
        response = client.post(
            "/auth/login",
            data={
                "username": "loginuser",
                "password": "loginpass123"
            }
        )
        assert response.status_code == 200
        data = response.json()
        assert "access_token" in data
        assert data["token_type"] == "bearer"

    def test_login_wrong_password(self):
        """Test login with wrong password"""
        client.post(
            "/auth/register",
            json={
                "username": "wrongpass",
                "email": "wrong@example.com",
                "password": "correctpassword"
            }
        )
        response = client.post(
            "/auth/login",
            data={
                "username": "wrongpass",
                "password": "wrongpassword"
            }
        )
        assert response.status_code == 401

    def test_login_nonexistent_user(self):
        """Test login with non-existent user"""
```

```
        response = client.post(
            "/auth/login",
            data={
                "username": "nonexistent",
                "password": "password123"
            }
        )
        assert response.status_code == 401

class TestUserAuthentication:
    """Test authentication and tokens"""

    def test_get_current_user(self):
        """Test getting current user info"""
        client.post(
            "/auth/register",
            json={
                "username": "authuser",
                "email": "auth@example.com",
                "password": "password123"
            }
        )
        login_resp = client.post(
            "/auth/login",
            data={"username": "authuser", "password": "password123"}
        )
        token = login_resp.json()["access_token"]

        response = client.get(
            "/auth/me",
            headers={"Authorization": f"Bearer {token}"}
        )
        assert response.status_code == 200
        data = response.json()
        assert data["username"] == "authuser"
        assert data["email"] == "auth@example.com"

    def test_get_current_user_no_token(self):
        """Test getting user without token"""
        response = client.get("/auth/me")
        assert response.status_code == 401

    def test_get_current_user_invalid_token(self):
        """Test with invalid token"""
        response = client.get(
            "/auth/me",
            headers={"Authorization": "Bearer invalid_token"}
        )
        assert response.status_code == 401

class TestPasswordSecurity:
    """Test password security"""

    def test_password_is_hashed(self):
```

```
        """Test that passwords are hashed"""
        response = client.post(
            "/auth/register",
            json={
                "username": "hashtest",
                "email": "hash@example.com",
                "password": "mypassword123"
            }
        )
        # Password should not be in response
        assert "password" not in response.json()
        assert "hashed_password" not in response.json()

    def test_password_verification(self):
        """Test password hashing and verification"""
        password = "testpassword123"
        hashed = auth.get_password_hash(password)

        # Correct password verifies
        assert auth.verify_password(password, hashed) is True

        # Wrong password doesn't verify
        assert auth.verify_password("wrongpassword", hashed) is False

    def test_different_hashes_for_same_password(self):
        """Test that same password produces different hashes"""
        password = "samepassword"
        hash1 = auth.get_password_hash(password)
        hash2 = auth.get_password_hash(password)

        # Hashes should be different
        assert hash1 != hash2

        # But both should verify
        assert auth.verify_password(password, hash1)
        assert auth.verify_password(password, hash2)

class TestUserProfile:
    """Test user profile features"""

    def test_user_has_created_at(self):
        """Test user has timestamp"""
        response = client.post(
            "/auth/register",
            json={
                "username": "timestamp",
                "email": "timestamp@example.com",
                "password": "password123"
            }
        )
        assert "created_at" in response.json()

    def test_user_is_active_by_default(self):
        """Test new users are active"""
```

```
        response = client.post(
            "/auth/register",
            json={
                "username": "activeuser",
                "email": "active@example.com",
                "password": "password123"
            }
        )
        assert response.json()["is_active"] is True

class TestUserRecipeRelationship:
    """Test user-recipe relationships"""

    def test_user_can_create_recipes(self):
        """Test user can create recipes"""
        # Register and login
        client.post(
            "/auth/register",
            json={
                "username": "recipeowner",
                "email": "recipeowner@example.com",
                "password": "password123"
            }
        )
        login_resp = client.post(
            "/auth/login",
            data={"username": "recipeowner", "password": "password123"}
        )
        token = login_resp.json()["access_token"]
        headers = {"Authorization": f"Bearer {token}"}

        # Create recipe
        response = client.post(
            "/recipes",
            json={
                "title": "User's Recipe",
                "ingredients": "test",
                "instructions": "test"
            },
            headers=headers
        )
        assert response.status_code == 200
        assert response.json()["title"] == "User's Recipe"

    def test_user_can_view_own_recipes(self):
        """Test user can view their own recipes"""
        # Register and login
        client.post(
            "/auth/register",
            json={
                "username": "viewer",
                "email": "viewer@example.com",
                "password": "password123"
            }
```

```
        )
        login_resp = client.post(
            "/auth/login",
            data={"username": "viewer", "password": "password123"}
        )
        token = login_resp.json()["access_token"]
        headers = {"Authorization": f"Bearer {token}"}

        # Create recipes
        for i in range(3):
            client.post(
                "/recipes",
                json={
                    "title": f"Recipe {i}",
                    "ingredients": "test",
                    "instructions": "test"
                },
                headers=headers
            )

        # Get user's recipes
        response = client.get("/recipes/my", headers=headers)
        assert response.status_code == 200
        assert len(response.json()) == 3

class TestMultipleUsers:
    """Test multiple user scenarios"""

    def test_users_have_separate_recipes(self):
        """Test users have separate recipe collections"""
        # User 1
        client.post(
            "/auth/register",
            json={
                "username": "user1sep",
                "email": "user1sep@example.com",
                "password": "password123"
            }
        )
        login1 = client.post(
            "/auth/login",
            data={"username": "user1sep", "password": "password123"}
        )
        token1 = login1.json()["access_token"]
        headers1 = {"Authorization": f"Bearer {token1}"}

        # User 2
        client.post(
            "/auth/register",
            json={
                "username": "user2sep",
                "email": "user2sep@example.com",
                "password": "password123"
            }
```

```
        )
        login2 = client.post(
            "/auth/login",
            data={"username": "user2sep", "password": "password123"}
        )
        token2 = login2.json()["access_token"]
        headers2 = {"Authorization": f"Bearer {token2}"}

        # User 1 creates recipes
        for i in range(2):
            client.post(
                "/recipes",
                json={
                    "title": f"User1 Recipe {i}",
                    "ingredients": "test",
                    "instructions": "test"
                },
                headers=headers1
            )

        # User 2 creates recipes
        for i in range(3):
            client.post(
                "/recipes",
                json={
                    "title": f"User2 Recipe {i}",
                    "ingredients": "test",
                    "instructions": "test"
                },
                headers=headers2
            )

        # Verify separate collections
        recipes1 = client.get("/recipes/my", headers=headers1).json()
        recipes2 = client.get("/recipes/my", headers=headers2).json()

        assert len(recipes1) == 2
        assert len(recipes2) == 3

    def test_users_cannot_modify_others_recipes(self):
        """Test users can't modify other users' recipes"""
        # User 1 creates recipe
        client.post(
            "/auth/register",
            json={
                "username": "owner",
                "email": "owner@example.com",
                "password": "password123"
            }
        )
        login1 = client.post(
            "/auth/login",
            data={"username": "owner", "password": "password123"}
        )
```

```
        token1 = login1.json()["access_token"]
        headers1 = {"Authorization": f"Bearer {token1}"}

        recipe_resp = client.post(
            "/recipes",
            json={
                "title": "Owner's Recipe",
                "ingredients": "test",
                "instructions": "test"
            },
            headers=headers1
        )
        recipe_id = recipe_resp.json()["id"]

        # User 2 tries to modify
        client.post(
            "/auth/register",
            json={
                "username": "hacker",
                "email": "hacker@example.com",
                "password": "password123"
            }
        )
        login2 = client.post(
            "/auth/login",
            data={"username": "hacker", "password": "password123"}
        )
        token2 = login2.json()["access_token"]
        headers2 = {"Authorization": f"Bearer {token2}"}

        response = client.put(
            f"/recipes/{recipe_id}",
            json={"title": "Hacked Title"},
            headers=headers2
        )
        assert response.status_code == 404  # Not found (no permission)
```

frontend

app.js

```
// Global variables
let currentUser = null;
let authToken = null;
let currentRecipes = [];
let currentPage = 0;
const recipesPerPage = 20;

// DOM elements
const loginBtn = document.getElementById('login-btn');
const logoutBtn = document.getElementById('logout-btn');
```

```
const loginModal = document.getElementById('login-modal');
const recipeModal = document.getElementById('recipe-modal');
const addRecipeModal = document.getElementById('add-recipe-modal');
const searchInput = document.getElementById('search-input');
const searchBtn = document.getElementById('search-btn');

// Initialize app
document.addEventListener('DOMContentLoaded', function() {
    initializeApp();
    setupEventListeners();
    checkAuthToken();
});

// Initialize app
function initializeApp() {
    showSection('home');
    loadRecipes();
}

// Setup event listeners
function setupEventListeners() {
    // Navigation
    document.querySelectorAll('.nav-link').forEach(link => {
        link.addEventListener('click', (e) => {
            e.preventDefault();
            const section = e.target.getAttribute('href').substring(1);
            showSection(section);
            loadSectionData(section);
            window.scrollTo({ top: 0, behavior: 'smooth' });
        });
    });

    // Authentication
    loginBtn.addEventListener('click', () => openModal('login-modal'));
    logoutBtn.addEventListener('click', logout);

    // Auth tabs
    document.querySelectorAll('.auth-tab').forEach(tab => {
        tab.addEventListener('click', (e) => {
            const tabType = e.target.dataset.tab;
            switchAuthTab(tabType);
        });
    });

    // Auth forms
    document.getElementById('login-form').addEventListener('submit', handleLogin);
    document.getElementById('register-form').addEventListener('submit',
handleRegister);
    document.getElementById('add-recipe-form').addEventListener('submit',
handleAddRecipe);

    // Search
    searchBtn.addEventListener('click', performSearch);
    searchInput.addEventListener('keypress', (e) => {
```

```
        if (e.key === 'Enter') performSearch();
    });

    // Add recipe button
    document.getElementById('add-recipe-btn').addEventListener('click', () => {
        openModal('add-recipe-modal');
    });

    // Load more button
    document.getElementById('load-more').addEventListener('click',
loadMoreRecipes);

    // Modal close buttons
    document.querySelectorAll('.close').forEach(closeBtn => {
        closeBtn.addEventListener('click', (e) => {
            const modal = e.target.closest('.modal');
            closeModal(modal.id);
        });
    });

    // Close modals when clicking outside
    window.addEventListener('click', (e) => {
        if (e.target.classList.contains('modal')) {
            closeModal(e.target.id);
        }
    });
}

// Authentication functions
function checkAuthToken() {
    authToken = localStorage.getItem('authToken');
    if (authToken) {
        fetchCurrentUser();
    }
}

async function fetchCurrentUser() {
    try {
        const response = await fetch('/auth/me', {
            headers: {
                'Authorization': `Bearer ${authToken}`
            }
        });

        if (response.ok) {
            currentUser = await response.json();
            updateUIForLoggedInUser();
        } else {
            localStorage.removeItem('authToken');
            authToken = null;
        }
    } catch (error) {
        console.error('Error fetching user:', error);
    }
```

```
}

function updateUIForLoggedInUser() {
    loginBtn.style.display = 'none';
    logoutBtn.style.display = 'block';
    document.getElementById('favorites-link').style.display = 'block';
    document.getElementById('my-recipes-link').style.display = 'block';
}

function updateUIForLoggedOutUser() {
    loginBtn.style.display = 'block';
    logoutBtn.style.display = 'none';
    document.getElementById('favorites-link').style.display = 'none';
    document.getElementById('my-recipes-link').style.display = 'none';
    currentUser = null;
}

async function handleLogin(e) {
    e.preventDefault();
    const username = document.getElementById('login-username').value;
    const password = document.getElementById('login-password').value;

    showLoading(true);
    try {
        const formData = new FormData();
        formData.append('username', username);
        formData.append('password', password);

        const response = await fetch('/auth/login', {
            method: 'POST',
            body: formData
        });

        const data = await response.json();

        if (response.ok) {
            authToken = data.access_token;
            localStorage.setItem('authToken', authToken);
            await fetchCurrentUser();
            closeModal('login-modal');
            showToast('Login successful!', 'success');
            document.getElementById('login-form').reset();
        } else {
            showToast(data.detail || 'Login failed', 'error');
        }
    } catch (error) {
        showToast('Network error. Please try again.', 'error');
        console.error('Login error:', error);
    } finally {
        showLoading(false);
    }
}

async function handleRegister(e) {
```

```
    e.preventDefault();
    const username = document.getElementById('register-username').value;
    const email = document.getElementById('register-email').value;
    const password = document.getElementById('register-password').value;

    showLoading(true);
    try {
        const response = await fetch('/auth/register', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ username, email, password })
        });

        const data = await response.json();

        if (response.ok) {
            showToast('Registration successful! Please login.', 'success');
            switchAuthTab('login');
            document.getElementById('register-form').reset();
        } else {
            showToast(data.detail || 'Registration failed', 'error');
        }
    } catch (error) {
        showToast('Network error. Please try again.', 'error');
        console.error('Register error:', error);
    } finally {
        showLoading(false);
    }
}

function logout() {
    localStorage.removeItem('authToken');
    authToken = null;
    updateUIForLoggedOutUser();
    showSection('home');
    showToast('Logged out successfully', 'success');
}

// Recipe functions
async function loadRecipes(reset = false) {
    if (reset) {
        currentRecipes = [];
        currentPage = 0;
    }

    showLoading(true);
    try {
        // Fetch local recipes
        const localResponse = await fetch(`/recipes?skip=${currentPage *
recipesPerPage}&limit=${recipesPerPage}`);

        if (!localResponse.ok) {
```

```
            throw new Error('Failed to load local recipes');
        }

        const localData = await localResponse.json();

        // Check if localData is an array or an object with a recipes property
        const localRecipes = Array.isArray(localData) ? localData :
(localData.recipes || []);

        let allRecipes = [...localRecipes];

        // On first load, fetch external recipes
        // On first load, fetch external recipes by category
if (currentPage === 0) {
    const categories = ['Pasta', 'Seafood', 'Chicken', 'Beef', 'Dessert',
'Vegetarian'];

    for (const category of categories) {
        try {
            const extResponse = await fetch(`/api/recipes/category?
c=${category}`);
            if (extResponse.ok) {
                const extData = await extResponse.json();
                if (extData.results && Array.isArray(extData.results)) {
                    allRecipes.push(...extData.results);
                }
            }
        } catch (err) {
            console.log(`Skipping ${category}:`, err.message);
        }
    }

    // Remove duplicates by ID
    const seen = new Set();
    allRecipes = allRecipes.filter(recipe => {
        const id = recipe.id;
        if (seen.has(id)) return false;
        seen.add(id);
        return true;
    });
}
        currentRecipes = reset ? allRecipes : [...currentRecipes, ...allRecipes];
        displayRecipes(currentRecipes, 'recipe-grid');
        currentPage++;

        // Manage load more button
        const loadMoreBtn = document.getElementById('load-more');
        if (loadMoreBtn) {
            loadMoreBtn.style.display = localRecipes.length < recipesPerPage ?
'none' : 'block';
        }

    } catch (error) {
        console.error('Load recipes error:', error);
```

```
            showToast('Error loading recipes: ' + error.message, 'error');
        } finally {
            showLoading(false);
        }
    }
    async function loadMyRecipes() {
        if (!authToken) {
            showToast('Please login to view your recipes', 'warning');
            return;
        }

        showLoading(true);
        try {
            const response = await fetch('/recipes/my', {
                headers: {
                    'Authorization': `Bearer ${authToken}`
                }
            });

            if (response.ok) {
                const recipes = await response.json();
                displayMyRecipes(recipes, 'my-recipes-grid');
            } else {
                const error = await response.json();
                showToast(error.detail || 'Error loading your recipes', 'error');
            }
        } catch (error) {
            showToast('Network error loading your recipes', 'error');
            console.error('Load my recipes error:', error);
        } finally {
            showLoading(false);
        }
    }

    async function loadFavorites() {
        if (!authToken) {
            showToast('Please login to view favorites', 'warning');
            return;
        }

        showLoading(true);
        try {
            const response = await fetch('/favorites', {
                headers: {
                    'Authorization': `Bearer ${authToken}`
                }
            });

            const favorites = await response.json();

            if (response.ok) {
                const favoriteRecipes = favorites.map(fav => fav.recipe);
                displayFavoriteRecipes(favoriteRecipes, 'favorites-grid');
            } else {
```

```
            showToast('Error loading favorites', 'error');
        }
    } catch (error) {
        showToast('Network error loading favorites', 'error');
        console.error('Load favorites error:', error);
    } finally {
        showLoading(false);
    }
}
function displayFavoriteRecipes(recipes, containerId) {
    const container = document.getElementById(containerId);
    container.innerHTML = '';

    if (recipes.length === 0) {
        container.innerHTML = '<div class="no-recipes"><p>No favorites yet. Add
recipes to favorites to see them here!</p></div>';
        return;
    }

    recipes.forEach(recipe => {
        const card = document.createElement('div');
        card.className = 'recipe-card';

        const imageUrl = recipe.image_url || 'https://via.placeholder.com/300x200?
text=No+Image';
        const prepTime = recipe.prep_time ? `${recipe.prep_time}min prep` : '';
        const cookTime = recipe.cook_time ? `${recipe.cook_time}min cook` : '';
        const timeInfo = [prepTime, cookTime].filter(t => t).join(' • ');

        card.innerHTML = `
            <img src="${imageUrl}" alt="${recipe.title}" class="recipe-image"
onerror="this.src='https://via.placeholder.com/300x200?text=No+Image'">
            <div class="recipe-content">
                <h3 class="recipe-title">${recipe.title}</h3>
                <p class="recipe-description">${recipe.description || 'No
description available'}</p>
                <div class="recipe-meta">
                    <span class="recipe-time">
                        <i class="fas fa-clock"></i>
                        ${timeInfo || 'Time not specified'}
                    </span>
                    <span class="recipe-difficulty">${recipe.difficulty ||
'medium'}</span>
                </div>
                <div class="recipe-actions">
                    <button onclick="removeFavorite(${recipe.id});
event.stopPropagation();" class="btn btn-danger">
                        <i class="fas fa-heart-broken"></i> Remove
                    </button>
                </div>
            </div>
        `;

        card.addEventListener('click', (e) => {
```

```
            if (!e.target.closest('button')) {
                showRecipeDetails(recipe.id);
            }
        });

        container.appendChild(card);
    });
}
// Original function for home/recipes/search tabs
function displayRecipes(recipes, containerId, showActions = false) {
    const container = document.getElementById(containerId);
    container.innerHTML = '';

    if (recipes.length === 0) {
        container.innerHTML = '<div class="no-recipes"><p>No recipes found.</p>
</div>';
        return;
    }

    recipes.forEach(recipe => {
        const recipeCard = createRecipeCard(recipe, showActions);
        container.appendChild(recipeCard);
    });
}

// Function for My Recipes tab with edit/delete buttons
function displayMyRecipes(recipes, containerId) {
    const container = document.getElementById(containerId);
    container.innerHTML = '';

    if (recipes.length === 0) {
        container.innerHTML = '<div class="no-recipes"><p>No recipes found. Create
your first recipe!</p></div>';
        return;
    }

    recipes.forEach(recipe => {
        const card = document.createElement('div');
        card.className = 'recipe-card';

        const imageUrl = recipe.image_url || 'https://via.placeholder.com/300x200?
text=No+Image';
        const prepTime = recipe.prep_time ? `${recipe.prep_time}min prep` : '';
        const cookTime = recipe.cook_time ? `${recipe.cook_time}min cook` : '';
        const timeInfo = [prepTime, cookTime].filter(t => t).join(' • ');

        card.innerHTML = `
            <img src="${imageUrl}" alt="${recipe.title}" class="recipe-image"
onerror="this.src='https://via.placeholder.com/300x200?text=No+Image'">
            <div class="recipe-content">
                <h3 class="recipe-title">${recipe.title}</h3>
                <p class="recipe-description">${recipe.description || 'No
description available'}</p>
                <div class="recipe-meta">
```

```
                    <span class="recipe-time"><i class="fas fa-clock"></i>
${timeInfo || 'Time not specified'}</span>
                    <span class="recipe-difficulty">${recipe.difficulty ||
'medium'}</span>
                </div>
                <div class="recipe-actions">
                    <button onclick="editRecipe(${recipe.id});
event.stopPropagation();" class="btn btn-secondary">
                        <i class="fas fa-edit"></i> Edit
                    </button>
                    <button onclick="deleteRecipe(${recipe.id});
event.stopPropagation();" class="btn btn-danger">
                        <i class="fas fa-trash"></i> Delete
                    </button>
                </div>
            </div>
        `;

        card.addEventListener('click', (e) => {
            if (!e.target.closest('button')) {
                showRecipeDetails(recipe.id);
            }
        });

        container.appendChild(card);
    });
}

async function removeFavorite(recipeId) {
    if (!authToken) {
        showToast('Please login', 'warning');
        return;
    }

    showLoading(true);
    try {
        const response = await fetch(`/favorites/${recipeId}`, {
            method: 'DELETE',
            headers: {
                'Authorization': `Bearer ${authToken}`
            }
        });

        if (response.ok) {
            showToast('Removed from favorites', 'success');
            loadFavorites(); // Refresh the list
        } else {
            showToast('Error removing from favorites', 'error');
        }
    } catch (error) {
        showToast('Network error', 'error');
        console.error('Remove favorite error:', error);
    } finally {
        showLoading(false);
```

```
    }
}



function createRecipeCard(recipe, showActions = false) {
    const card = document.createElement('div');
    card.className = 'recipe-card';
    const isExternal = recipe.external || String(recipe.id).startsWith('ext_');

    const imageUrl = recipe.image_url || 'https://via.placeholder.com/300x200?
text=No+Image';
    const prepTime = recipe.prep_time ? `${recipe.prep_time}min prep` : '';
    const cookTime = recipe.cook_time ? `${recipe.cook_time}min cook` : '';
    const timeInfo = [prepTime, cookTime].filter(t => t).join(' • ');

    card.innerHTML = `
        <img src="${imageUrl}" alt="${recipe.title}" class="recipe-image"
onerror="this.src='https://via.placeholder.com/300x200?text=No+Image'">
        <div class="recipe-content">
            <h3 class="recipe-title">${recipe.title} ${isExternal ? '🌐' : ''}
</h3>
            <p class="recipe-description">${recipe.description || 'No description
available'}</p>
            <div class="recipe-meta">
                <span class="recipe-time">
                    <i class="fas fa-clock"></i>
                    ${timeInfo || 'Time not specified'}
                </span>
                <span class="recipe-difficulty">${recipe.difficulty || 'easy'}
</span>
            </div>
            <div class="recipe-actions">
                <button class="btn btn-favorite"
onclick="handleFavoriteClick('${recipe.id}', ${isExternal}, event)">
                    <i class="fas fa-heart"></i> ${isExternal ? 'Save' : ''}
                </button>
                ${showActions && !isExternal ? `
                    <button onclick="editRecipe(${recipe.id});
event.stopPropagation();" class="btn btn-secondary">
                        <i class="fas fa-edit"></i> Edit
                    </button>
                    <button onclick="deleteRecipe(${recipe.id});
event.stopPropagation();" class="btn btn-danger">
                        <i class="fas fa-trash"></i> Delete
                    </button>
                ` : ''}
            </div>
        </div>
    `;

    card.addEventListener('click', (e) => {
        if (!e.target.closest('button')) {
            showRecipeDetails(recipe.id);
```

```
        }
    });
    if (isExternal) {
        card.dataset.recipeData = JSON.stringify(recipe);
    }
    return card;
}

async function handleFavoriteClick(recipeId, isExternal) {
    if (!authToken) {
        showToast('Please login to add favorites', 'warning');
        return;
    }

    showLoading(true);
    try {
        if (isExternal) {
            // Get the full recipe data from the card
            const card = event.target.closest('.recipe-card');
            const recipeData = JSON.parse(card.dataset.recipeData);

            // First save to database
            showToast('Saving recipe to your collection...', 'info');
            const savedId = await saveExternalRecipe(recipeData);

            if (savedId) {
                // Then add to favorites
                const response = await fetch(`/favorites/${savedId}`, {
                    method: 'POST',
                    headers: { 'Authorization': `Bearer ${authToken}` }
                });

                if (response.ok) {
                    showToast('Recipe saved and added to favorites!', 'success');
                } else {
                    showToast('Recipe saved but could not add to favorites',
'warning');
                }
            } else {
                showToast('Error saving recipe', 'error');
            }
        } else {
            // Regular favorite toggle for your recipes
            await toggleFavorite(recipeId);
        }
    } catch (error) {
        showToast('Error processing favorite', 'error');
        console.error('Favorite error:', error);
    } finally {
        showLoading(false);
    }
}

async function showRecipeDetails(recipeId) {
```

```
    showLoading(true);
    try {
        // Check if external recipe (starts with "ext_")
        if (String(recipeId).startsWith('ext_')) {
            const mealId = String(recipeId).replace('ext_', '');
            const response = await
fetch(`https://www.themealdb.com/api/json/v1/1/lookup.php?i=${mealId}`);
            const data = await response.json();
            const recipe = data.meals[0];

            // Build ingredients list
            const ingredients = [];
            for (let i = 1; i <= 20; i++) {
                const ingredient = recipe[`strIngredient${i}`];
                const measure = recipe[`strMeasure${i}`];
                if (ingredient && ingredient.trim()) {
                    ingredients.push(`${measure} ${ingredient}`.trim());
                }
            }

            const externalRecipe = {
                title: recipe.strMeal,
                image_url: recipe.strMealThumb,
                description: `${recipe.strCategory} - ${recipe.strArea}`,
                ingredients: ingredients.join('\n'),
                instructions: recipe.strInstructions,
                source_url: recipe.strYoutube
            };

            displayRecipeDetails(externalRecipe);
            openModal('recipe-modal');
        } else {
            // Local recipe
            const response = await fetch(`/recipes/${recipeId}`);
            const recipe = await response.json();

            if (response.ok) {
                displayRecipeDetails(recipe);
                openModal('recipe-modal');
            } else {
                showToast('Error loading recipe details', 'error');
            }
        }
    } catch (error) {
        showToast('Network error loading recipe details', 'error');
        console.error('Recipe details error:', error);
    } finally {
        showLoading(false);
    }
}
function displayRecipeDetails(recipe) {
    const detailsContainer = document.getElementById('recipe-details');
    const imageUrl = recipe.image_url || 'https://via.placeholder.com/400x250?
text=No+Image';
```

```
    // Format ingredients (assuming they're stored as comma-separated or line-
separated)
    const ingredients = recipe.ingredients.split(/[,\n]/).map(ing =>
ing.trim()).filter(ing => ing);
    const ingredientsList = ingredients.map(ing => `<li>${ing}</li>`).join('');

    // Format instructions
    const instructions = recipe.instructions.replace(/\n/g, '<br>');

    detailsContainer.innerHTML = `
        <div class="recipe-detail">
            <div class="recipe-detail-header">
                <h2 class="recipe-detail-title">${recipe.title}</h2>
                <img src="${imageUrl}" alt="${recipe.title}" class="recipe-detail-
image" onerror="this.src='https://via.placeholder.com/400x250?text=No+Image'">
                <div class="recipe-detail-meta">
                    ${recipe.prep_time ? `<span><i class="fas fa-clock"></i> Prep:
${recipe.prep_time}min</span>` : ''}
                    ${recipe.cook_time ? `<span><i class="fas fa-fire"></i> Cook:
${recipe.cook_time}min</span>` : ''}
                    <span><i class="fas fa-users"></i> Serves: ${recipe.servings
|| 1}</span>
                    <span><i class="fas fa-signal"></i> ${recipe.difficulty ||
'easy'}</span>
                </div>
            </div>

            ${recipe.description ? `
                <div class="recipe-detail-section">
                    <h4>Description</h4>
                    <p>${recipe.description}</p>
                </div>
            ` : ''}

            <div class="recipe-detail-section">
                <h4>Ingredients</h4>
                <ul class="ingredients-list">
                    ${ingredientsList}
                </ul>
            </div>

            <div class="recipe-detail-section">
                <h4>Instructions</h4>
                <div class="instructions-content">
                    ${instructions}
                </div>
            </div>
        </div>
    `;
}

async function handleAddRecipe(e) {
    e.preventDefault();
```

```
    if (!authToken) {
        showToast('Please login to add recipes', 'warning');
        return;
    }

    const recipeData = {
        title: document.getElementById('recipe-title').value,
        description: document.getElementById('recipe-description').value,
        ingredients: document.getElementById('recipe-ingredients').value,
        instructions: document.getElementById('recipe-instructions').value,
        prep_time: parseInt(document.getElementById('recipe-prep-time').value) ||
null,
        cook_time: parseInt(document.getElementById('recipe-cook-time').value) ||
null,
        servings: parseInt(document.getElementById('recipe-servings').value) || 1,
        difficulty: document.getElementById('recipe-difficulty').value,
        image_url: document.getElementById('recipe-image-url').value || null
    };

    showLoading(true);
    try {
        const response = await fetch('/recipes', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${authToken}`
            },
            body: JSON.stringify(recipeData)
        });

        const data = await response.json();

        if (response.ok) {
            showToast('Recipe added successfully!', 'success');
            closeModal('add-recipe-modal');
            document.getElementById('add-recipe-form').reset();
            // Refresh recipes if we're on the my-recipes page
            const activeSection = document.querySelector('.section.active');
            if (activeSection.id === 'my-recipes') {
                loadMyRecipes();
            }
        } else {
            showToast(data.detail || 'Error adding recipe', 'error');
        }
    } catch (error) {
        showToast('Network error adding recipe', 'error');
        console.error('Add recipe error:', error);
    } finally {
        showLoading(false);
    }
}

async function deleteRecipe(recipeId) {
```

```
    if (!confirm('Are you sure you want to delete this recipe?')) {
        return;
    }

    showLoading(true);
    try {
        const response = await fetch(`/recipes/${recipeId}`, {
            method: 'DELETE',
            headers: {
                'Authorization': `Bearer ${authToken}`
            }
        });

        if (response.ok) {
            showToast('Recipe deleted successfully', 'success');
            loadMyRecipes(); // Refresh the list
        } else {
            const data = await response.json();
            showToast(data.detail || 'Error deleting recipe', 'error');
        }
    } catch (error) {
        showToast('Network error deleting recipe', 'error');
        console.error('Delete recipe error:', error);
    } finally {
        showLoading(false);
    }
}

async function toggleFavorite(recipeId) {
    if (!authToken) {
        showToast('Please login to add favorites', 'warning');
        return;
    }

    try {
        // Try to add to favorites first
        const response = await fetch(`/favorites/${recipeId}`, {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${authToken}`
            }
        });

        if (response.ok) {
            showToast('Added to favorites!', 'success');
        } else if (response.status === 400) {
            // If already exists, try to remove
            const removeResponse = await fetch(`/favorites/${recipeId}`, {
                method: 'DELETE',
                headers: {
                    'Authorization': `Bearer ${authToken}`
                }
            });
```

```
            if (removeResponse.ok) {
                showToast('Removed from favorites', 'success');
            }
        } else {
            showToast('Error updating favorites', 'error');
        }
    } catch (error) {
        showToast('Network error updating favorites', 'error');
        console.error('Toggle favorite error:', error);
    }
}
async function saveExternalRecipe(externalRecipe) {
    if (!authToken) {
        showToast('Please login to save recipes', 'warning');
        return null;
    }

    try {
        // Ensure ingredients is a string
        let ingredientsStr = '';
        if (typeof externalRecipe.ingredients === 'string') {
            ingredientsStr = externalRecipe.ingredients;
        } else if (Array.isArray(externalRecipe.ingredients)) {
            ingredientsStr = externalRecipe.ingredients.join('\n');
        }

        // Ensure instructions is a string
        let instructionsStr = externalRecipe.instructions || 'No instructions
provided';
        if (typeof instructionsStr !== 'string') {
            instructionsStr = String(instructionsStr);
        }

        const recipeData = {
            title: externalRecipe.title || 'Untitled Recipe',
            description: externalRecipe.description || 'Recipe from TheMealDB',
            ingredients: ingredientsStr || 'No ingredients listed',
            instructions: instructionsStr,
            image_url: externalRecipe.image_url || null,
            difficulty: 'medium',
            servings: 4,
            prep_time: null,
            cook_time: null
        };

        console.log('Saving recipe:', recipeData); // Debug log

        const response = await fetch('/recipes', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': `Bearer ${authToken}`
            },
            body: JSON.stringify(recipeData)
```

```
        });

        if (response.ok) {
            const savedRecipe = await response.json();
            return savedRecipe.id;
        } else {
            const error = await response.json();
            console.error('Save recipe error:', error);
            showToast('Error: ' + (error.detail || 'Could not save recipe'),
'error');
            return null;
        }
    } catch (error) {
        console.error('Error saving external recipe:', error);
        showToast('Network error saving recipe', 'error');
        return null;
    }
}

async function performSearch() {
    const query = searchInput.value.trim().toLowerCase();
    if (!query) {
        showToast('Please enter a search term', 'warning');
        return;
    }

    showLoading(true);
    try {
        // Categories that should use category search instead of ingredient search
        const categories = [
            'pasta', 'dessert', 'seafood', 'vegetarian', 'vegan', 'breakfast',
            'beef', 'chicken', 'lamb', 'pork', 'side', 'starter', 'goat',
            'miscellaneous', 'soup', 'curry', 'salad'
        ];

        const isCategory = categories.includes(query);

        // Search local database
        const localPromise = fetch(`/recipes/search?
q=${encodeURIComponent(query)}&limit=20`).catch(() => null);

        // Search external API - use category or ingredient endpoint
        let externalPromise;
        if (isCategory) {
            externalPromise = fetch(`/api/recipes/category?
c=${encodeURIComponent(query)}`).catch(() => null);
        } else {
            externalPromise = fetch(`/api/recipes/external?
q=${encodeURIComponent(query)}`).catch(() => null);
        }

        const [localResponse, externalResponse] = await Promise.all([localPromise,
externalPromise]);
```

```
        let localRecipes = [];
        let externalRecipes = [];

        if (localResponse && localResponse.ok) {
            localRecipes = await localResponse.json();
        }

        if (externalResponse && externalResponse.ok) {
            const externalData = await externalResponse.json();
            externalRecipes = externalData.results || [];
        }

        const allRecipes = [...localRecipes, ...externalRecipes];

        if (allRecipes.length > 0) {
            displayRecipes(allRecipes, 'recipe-grid');
            showSection('recipes');
            showToast(`Found ${allRecipes.length} recipes (${localRecipes.length}
yours, ${externalRecipes.length} from web)`, 'success');
        } else {
            displayRecipes([], 'recipe-grid');
            showSection('recipes');
            showToast('No recipes found. Try ingredients like "chicken", "milk" or
categories like "pasta", "dessert"', 'warning');
        }
    } catch (error) {
        showToast('Network error searching recipes', 'error');
        console.error('Search error:', error);
    } finally {
        showLoading(false);
    }
}

window.quickSearch = function(term) {
    searchInput.value = term;
    performSearch();
};
// UI Helper functions
function showSection(sectionId) {
    document.querySelectorAll('.section').forEach(section => {
        section.classList.remove('active');
    });
    document.getElementById(sectionId).classList.add('active');
}

function loadSectionData(sectionId) {
    switch (sectionId) {
        case 'recipes':
            loadRecipes(true);
            break;
        case 'favorites':
            loadFavorites();
            break;
        case 'my-recipes':
```

```
            loadMyRecipes();
            break;
    }
}

function loadMoreRecipes() {
    loadRecipes();
}

function openModal(modalId) {
    document.getElementById(modalId).style.display = 'block';
}

function closeModal(modalId) {
    document.getElementById(modalId).style.display = 'none';
}

function switchAuthTab(tabType) {
    document.querySelectorAll('.auth-tab').forEach(tab => {
        tab.classList.remove('active');
    });
    document.querySelector(`[data-tab="${tabType}"]`).classList.add('active');

    document.querySelectorAll('.auth-form').forEach(form => {
        form.style.display = 'none';
    });
    document.getElementById(`${tabType}-form`).style.display = 'block';
}

function showLoading(show) {
    document.getElementById('loading').style.display = show ? 'flex' : 'none';
}

function showToast(message, type = 'info') {
    const toast = document.createElement('div');
    toast.className = `toast ${type}`;
    toast.textContent = message;

    const container = document.getElementById('toast-container');
    container.appendChild(toast);

    setTimeout(() => {
        toast.remove();
    }, 4000);
}

async function editRecipe(recipeId) {
    if (!authToken) {
        showToast('Please login to edit recipes', 'warning');
        return;
    }

    try {
        // Get the recipe details first
```

```
        const response = await fetch(`/recipes/${recipeId}`, {
            headers: {
                'Authorization': `Bearer ${authToken}`
            }
        });
        const recipe = await response.json();

        if (!response.ok) {
            showToast(recipe.detail || 'Error loading recipe', 'error');
            return;
        }

        // Prefill form with existing recipe data
        document.getElementById('recipe-title').value = recipe.title;
        document.getElementById('recipe-description').value = recipe.description
|| '';
        document.getElementById('recipe-ingredients').value = recipe.ingredients
|| '';
        document.getElementById('recipe-instructions').value = recipe.instructions
|| '';
        document.getElementById('recipe-prep-time').value = recipe.prep_time ||
'';
        document.getElementById('recipe-cook-time').value = recipe.cook_time ||
'';
        document.getElementById('recipe-servings').value = recipe.servings || 1;
        document.getElementById('recipe-difficulty').value = recipe.difficulty ||
'easy';
        document.getElementById('recipe-image-url').value = recipe.image_url ||
'';

        // Open the modal
        openModal('add-recipe-modal');

        // Replace form submit handler temporarily
        const form = document.getElementById('add-recipe-form');
        form.onsubmit = async function(e) {
            e.preventDefault();

            const updatedData = {
                title: document.getElementById('recipe-title').value,
                description: document.getElementById('recipe-description').value,
                ingredients: document.getElementById('recipe-ingredients').value,
                instructions: document.getElementById('recipe-
instructions').value,
                prep_time: parseInt(document.getElementById('recipe-prep-
time').value) || null,
                cook_time: parseInt(document.getElementById('recipe-cook-
time').value) || null,
                servings: parseInt(document.getElementById('recipe-
servings').value) || 1,
                difficulty: document.getElementById('recipe-difficulty').value,
                image_url: document.getElementById('recipe-image-url').value ||
null
            };
```

```
            try {
                const updateResponse = await fetch(`/recipes/${recipeId}`, {
                    method: 'PUT',
                    headers: {
                        'Content-Type': 'application/json',
                        'Authorization': `Bearer ${authToken}`
                    },
                    body: JSON.stringify(updatedData)
                });

                if (updateResponse.ok) {
                    showToast('Recipe updated successfully!', 'success');
                    closeModal('add-recipe-modal');
                    loadMyRecipes(); // refresh list
                } else {
                    const data = await updateResponse.json();
                    showToast(data.detail || 'Error updating recipe', 'error');
                }
            } catch (error) {
                showToast('Network error updating recipe', 'error');
                console.error('Edit recipe error:', error);
            }
        };
    } catch (error) {
        showToast('Error loading recipe for edit', 'error');
        console.error('Edit recipe fetch error:', error);
    }
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Recipe Finder</title>
    <link rel="stylesheet" href="/static/styles.css">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css" rel="stylesheet">
</head>
<body>
    <!-- Navigation -->
    <nav class="navbar">
        <div class="nav-container">
            <div class="nav-brand">
                <i class="fas fa-utensils"></i>
                <span>Recipe Finder</span>
            </div>
```

```
            <div class="nav-menu" id="nav-menu">
                <a href="#home" class="nav-link">Home</a>
                <a href="#recipes" class="nav-link">Recipes</a>
                <a href="#favorites" class="nav-link" id="favorites-link"
style="display: none;">Favorites</a>
                <a href="#my-recipes" class="nav-link" id="my-recipes-link"
style="display: none;">My Recipes</a>
                <button class="nav-btn" id="login-btn">Login</button>
                <button class="nav-btn" id="logout-btn" style="display:
none;">Logout</button>
            </div>
        </div>
    </nav>

    <!-- Main Content -->
    <main class="main-content">
        <!-- Hero Section -->
        <section id="home" class="section active">
            <div class="hero">
                <h1>Find Your Perfect Recipe</h1>
                <p>Discover, save, and share amazing recipes from around the
world</p>
                <div class="search-container">
                    <input type="text" id="search-input" placeholder="Search for
recipes..." class="search-input">
                    <button id="search-btn" class="search-btn">
                        <i class="fas fa-search"></i>
                    </button>
                </div>

                <!-- Quick Search Suggestions -->
                <div class="quick-search">
                    <div class="quick-search-section">
                        <h3><i class="fas fa-list"></i> Browse by Category</h3>
                        <div class="quick-search-buttons">
                            <button class="quick-btn"
onclick="quickSearch('chicken')">Chicken</button>
                            <button class="quick-btn"
onclick="quickSearch('beef')">Beef</button>
                            <button class="quick-btn"
onclick="quickSearch('seafood')">Seafood</button>
                            <button class="quick-btn"
onclick="quickSearch('pasta')">Pasta</button>
                            <button class="quick-btn"
onclick="quickSearch('dessert')">Dessert</button>
                            <button class="quick-btn"
onclick="quickSearch('vegetarian')">Vegetarian</button>
                            <button class="quick-btn"
onclick="quickSearch('vegan')">Vegan</button>
                            <button class="quick-btn"
onclick="quickSearch('breakfast')">Breakfast</button>
                        </div>
                    </div>
```

```
                    <div class="quick-search-section">
                        <h3><i class="fas fa-carrot"></i> Search by
Ingredient</h3>
                        <div class="quick-search-buttons">
                            <button class="quick-btn"
onclick="quickSearch('milk')">Milk</button>
                            <button class="quick-btn"
onclick="quickSearch('cheese')">Cheese</button>
                            <button class="quick-btn"
onclick="quickSearch('egg')">Eggs</button>
                            <button class="quick-btn"
onclick="quickSearch('tomato')">Tomato</button>
                            <button class="quick-btn"
onclick="quickSearch('potato')">Potato</button>
                            <button class="quick-btn"
onclick="quickSearch('rice')">Rice</button>
                            <button class="quick-btn"
onclick="quickSearch('garlic')">Garlic</button>
                            <button class="quick-btn"
onclick="quickSearch('lemon')">Lemon</button>
                        </div>
                    </div>
                </div>
            </div>
        </section>

        <!-- Recipes Section -->
        <section id="recipes" class="section">
            <div class="container">
                <h2>All Recipes</h2>
                <div class="recipe-grid" id="recipe-grid">
                    <!-- Recipes will be loaded here -->
                </div>
                <button id="load-more" class="load-more-btn">Load More</button>
            </div>
        </section>

        <!-- Favorites Section -->
        <section id="favorites" class="section">
            <div class="container">
                <h2>My Favorites</h2>
                <div class="recipe-grid" id="favorites-grid">
                    <!-- Favorites will be loaded here -->
                </div>
            </div>
        </section>

        <!-- My Recipes Section -->
        <section id="my-recipes" class="section">
            <div class="container">
                <div class="section-header">
                    <h2>My Recipes</h2>
                    <button id="add-recipe-btn" class="btn btn-primary">
                        <i class="fas fa-plus"></i> Add Recipe
```

```
                    </button>
                </div>
                <div class="recipe-grid" id="my-recipes-grid">
                    <!-- User's recipes will be loaded here -->
                </div>
            </div>
        </section>
    </main>

    <!-- Login Modal -->
    <div id="login-modal" class="modal">
        <div class="modal-content">
            <span class="close">&times;</span>
            <div class="auth-container">
                <div class="auth-tabs">
                    <button class="auth-tab active" data-
tab="login">Login</button>
                    <button class="auth-tab" data-tab="register">Register</button>
                </div>

                <!-- Login Form -->
                <form id="login-form" class="auth-form">
                    <h3>Login to Recipe Finder</h3>
                    <div class="form-group">
                        <input type="text" id="login-username"
placeholder="Username" required>
                    </div>
                    <div class="form-group">
                        <input type="password" id="login-password"
placeholder="Password" required>
                    </div>
                    <button type="submit" class="btn btn-primary">Login</button>
                </form>

                <!-- Register Form -->
                <form id="register-form" class="auth-form" style="display: none;">
                    <h3>Create an Account</h3>
                    <div class="form-group">
                        <input type="text" id="register-username"
placeholder="Username" required>
                    </div>
                    <div class="form-group">
                        <input type="email" id="register-email"
placeholder="Email" required>
                    </div>
                    <div class="form-group">
                        <input type="password" id="register-password"
placeholder="Password (min 6 characters)" required>
                    </div>
                    <button type="submit" class="btn btn-
primary">Register</button>
                </form>
            </div>
        </div>
```

```
    </div>

    <!-- Recipe Modal -->
    <div id="recipe-modal" class="modal">
        <div class="modal-content large">
            <span class="close">&times;</span>
            <div id="recipe-details">
                <!-- Recipe details will be loaded here -->
            </div>
        </div>
    </div>

    <!-- Add Recipe Modal -->
    <div id="add-recipe-modal" class="modal">
        <div class="modal-content large">
            <span class="close">&times;</span>
            <form id="add-recipe-form">
                <h3>Add New Recipe</h3>
                <div class="form-group">
                    <label for="recipe-title">Recipe Title *</label>
                    <input type="text" id="recipe-title" required>
                </div>
                <div class="form-group">
                    <label for="recipe-description">Description</label>
                    <textarea id="recipe-description" rows="3"></textarea>
                </div>
                <div class="form-group">
                    <label for="recipe-ingredients">Ingredients *</label>
                    <textarea id="recipe-ingredients" rows="5" placeholder="List
ingredients, one per line or separated by commas" required></textarea>
                </div>
                <div class="form-group">
                    <label for="recipe-instructions">Instructions *</label>
                    <textarea id="recipe-instructions" rows="6" placeholder="Step-
by-step cooking instructions" required></textarea>
                </div>
                <div class="form-row">
                    <div class="form-group">
                        <label for="recipe-prep-time">Prep Time (minutes)</label>
                        <input type="number" id="recipe-prep-time" min="0">
                    </div>
                    <div class="form-group">
                        <label for="recipe-cook-time">Cook Time (minutes)</label>
                        <input type="number" id="recipe-cook-time" min="0">
                    </div>
                    <div class="form-group">
                        <label for="recipe-servings">Servings</label>
                        <input type="number" id="recipe-servings" min="1"
value="1">
                    </div>
                </div>
                <div class="form-group">
                    <label for="recipe-difficulty">Difficulty</label>
                    <select id="recipe-difficulty">
```

```
                        <option value="easy">Easy</option>
                        <option value="medium">Medium</option>
                        <option value="hard">Hard</option>
                    </select>
                </div>
                <div class="form-group">
                    <label for="recipe-image-url">Image URL (optional)</label>
                    <input type="url" id="recipe-image-url"
placeholder="https://example.com/image.jpg">
                </div>
                <div class="form-actions">
                    <button type="button" class="btn btn-secondary"
onclick="closeModal('add-recipe-modal')">Cancel</button>
                    <button type="submit" class="btn btn-primary">Save
Recipe</button>
                </div>
            </form>
        </div>
    </div>

    <!-- Loading Spinner -->
    <div id="loading" class="loading" style="display: none;">
        <div class="spinner"></div>
    </div>

    <!-- Toast Notifications -->
    <div id="toast-container" class="toast-container"></div>

    <script src="/static/app.js"></script>
</body>
</html>
```

styles.css

```
/* Reset and Base Styles */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    line-height: 1.6;
    color: #333;
    background-color: #f8f9fa;
    padding-top: 100px; /* Increased to ensure content stays below navbar */
}

.container {
    max-width: 1200px;
```

```
    margin: 0 auto;
    padding: 0 20px;
}

/* Navigation */
.navbar {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    padding: 1rem 0;
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    z-index: 1000;
    box-shadow: 0 2px 10px rgba(0,0,0,0.1);
}

.nav-container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 0 20px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

.nav-brand {
    display: flex;
    align-items: center;
    font-size: 1.5rem;
    font-weight: bold;
}

.nav-brand i {
    margin-right: 0.5rem;
    font-size: 1.8rem;
}

.nav-menu {
    display: flex;
    align-items: center;
    gap: 2rem;
}

.nav-link {
    color: white;
    text-decoration: none;
    padding: 0.5rem 1rem;
    border-radius: 5px;
    transition: background-color 0.3s;
}

.nav-link:hover {
    background-color: rgba(255,255,255,0.1);
```

```
}

.nav-btn {
    background: rgba(255,255,255,0.2);
    color: white;
    border: none;
    padding: 0.5rem 1rem;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}

.nav-btn:hover {
    background: rgba(255,255,255,0.3);
}

/* Main Content */
.main-content {
    margin-top: 0; /* Changed from 80px since body now has padding-top */
    min-height: calc(100vh - 80px);
}

.section {
    display: none;
    padding: 2rem 0;
    min-height: calc(100vh - 80px);
}

.section.active {
    display: block;
}

.section > .container {
    padding-top: 1rem; /* Extra padding for all section containers */
}

.section > .container > h2:first-child {
    padding-top: 1rem; /* Extra padding for standalone h2 titles */
}

/* Hero Section */
.hero {
    text-align: center;
    padding: 4rem 2rem;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
}

.hero h1 {
    font-size: 3rem;
    margin-bottom: 1rem;
    animation: fadeInUp 1s ease-out;
}
```

```
.hero p {
    font-size: 1.2rem;
    margin-bottom: 2rem;
    animation: fadeInUp 1s ease-out 0.2s both;
}

.search-container {
    max-width: 600px;
    margin: 0 auto;
    display: flex;
    gap: 0.5rem;
    animation: fadeInUp 1s ease-out 0.4s both;
}

.search-input {
    flex: 1;
    padding: 1rem;
    border: none;
    border-radius: 10px;
    font-size: 1rem;
    outline: none;
}

.search-btn {
    background: #ff6b6b;
    color: white;
    border: none;
    padding: 1rem 1.5rem;
    border-radius: 10px;
    cursor: pointer;
    font-size: 1rem;
    transition: background-color 0.3s;
}

.search-btn:hover {
    background: #ff5252;
}

/* Recipe Grid */
.section-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 2rem;
    padding-top: 2rem; /* Increased padding to ensure it's below navbar */
}

.recipe-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
    gap: 2rem;
    margin-bottom: 2rem;
}
```

```css
.recipe-card {
    background: white;
    border-radius: 15px;
    overflow: hidden;
    box-shadow: 0 5px 15px rgba(0,0,0,0.1);
    transition: transform 0.3s, box-shadow 0.3s;
    cursor: pointer;
}

.recipe-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 10px 25px rgba(0,0,0,0.15);
}

.recipe-image {
    width: 100%;
    height: 200px;
    object-fit: cover;
    background: linear-gradient(45deg, #f0f0f0, #e0e0e0);
}

.recipe-content {
    padding: 1.5rem;
}

.recipe-title {
    font-size: 1.2rem;
    font-weight: bold;
    margin-bottom: 0.5rem;
    color: #333;
}

.recipe-description {
    color: #666;
    margin-bottom: 1rem;
    display: -webkit-box;
    -webkit-line-clamp: 2;
    -webkit-box-orient: vertical;
    overflow: hidden;
}

.recipe-meta {
    display: flex;
    justify-content: space-between;
    align-items: center;
    font-size: 0.9rem;
    color: #888;
}

.recipe-time {
    display: flex;
    align-items: center;
    gap: 0.25rem;
}
```

```
.recipe-actions {
    display: flex;
    gap: 0.5rem;
    margin-top: 1rem;
}

.btn {
    padding: 0.5rem 1rem;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 0.9rem;
    transition: all 0.3s;
    text-decoration: none;
    display: inline-flex;
    align-items: center;
    gap: 0.5rem;
}

.btn-primary {
    background: #667eea;
    color: white;
}

.btn-primary:hover {
    background: #5a67d8;
}

.btn-secondary {
    background: #e2e8f0;
    color: #4a5568;
}

.btn-secondary:hover {
    background: #cbd5e0;
}

.btn-danger {
    background: #ff6b6b;
    color: white;
}

.btn-danger:hover {
    background: #ff5252;
}

.btn-favorite {
    background: #ff6b6b;
    color: white;
}

.btn-favorite.active {
    background: #ff1744;
```

```
}

/* Load More Button */
.load-more-btn {
    display: block;
    margin: 2rem auto;
    padding: 1rem 2rem;
    background: #667eea;
    color: white;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    font-size: 1rem;
    transition: background-color 0.3s;
}

.load-more-btn:hover {
    background: #5a67d8;
}

/* Modal Styles */
.modal {
    display: none;
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0,0,0,0.5);
    z-index: 2000;
    animation: fadeIn 0.3s ease-out;
}

.modal-content {
    background: white;
    margin: 5% auto;
    padding: 2rem;
    border-radius: 15px;
    max-width: 500px;
    position: relative;
    animation: slideIn 0.3s ease-out;
    max-height: 80vh;
    overflow-y: auto;
}

.modal-content.large {
    max-width: 800px;
}

.close {
    position: absolute;
    top: 1rem;
    right: 1.5rem;
    font-size: 2rem;
```

```
    cursor: pointer;
    color: #aaa;
    transition: color 0.3s;
}

.close:hover {
    color: #333;
}

/* Auth Forms */
.auth-tabs {
    display: flex;
    margin-bottom: 2rem;
    border-bottom: 1px solid #e2e8f0;
}

.auth-tab {
    flex: 1;
    padding: 1rem;
    border: none;
    background: none;
    cursor: pointer;
    transition: all 0.3s;
}

.auth-tab.active {
    border-bottom: 2px solid #667eea;
    color: #667eea;
    font-weight: bold;
}

.auth-form h3 {
    text-align: center;
    margin-bottom: 1.5rem;
    color: #333;
}

.form-group {
    margin-bottom: 1rem;
}

.form-group label {
    display: block;
    margin-bottom: 0.5rem;
    font-weight: 500;
    color: #333;
}

.form-group input,
.form-group textarea,
.form-group select {
    width: 100%;
    padding: 0.75rem;
    border: 1px solid #e2e8f0;
```

```
    border-radius: 5px;
    font-size: 1rem;
    transition: border-color 0.3s;
}

.form-group input:focus,
.form-group textarea:focus,
.form-group select:focus {
    outline: none;
    border-color: #667eea;
}

.form-row {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
    gap: 1rem;
}

.form-actions {
    display: flex;
    gap: 1rem;
    justify-content: flex-end;
    margin-top: 2rem;
}

/* Recipe Details */
.recipe-detail {
    display: grid;
    gap: 2rem;
}

.recipe-detail-header {
    text-align: center;
}

.recipe-detail-title {
    font-size: 2rem;
    margin-bottom: 1rem;
    color: #333;
}

.recipe-detail-image {
    width: 100%;
    max-width: 400px;
    height: 250px;
    object-fit: cover;
    border-radius: 10px;
    margin: 0 auto 1rem;
}

.recipe-detail-meta {
    display: flex;
    justify-content: center;
    gap: 2rem;
```

```
    margin-bottom: 1rem;
    font-size: 0.9rem;
    color: #666;
}

.recipe-detail-section {
    margin-bottom: 2rem;
}

.recipe-detail-section h4 {
    font-size: 1.3rem;
    margin-bottom: 1rem;
    color: #333;
    border-bottom: 2px solid #667eea;
    padding-bottom: 0.5rem;
}

.ingredients-list {
    list-style: none;
    padding: 0;
}

.ingredients-list li {
    padding: 0.5rem 0;
    border-bottom: 1px solid #f0f0f0;
}

.ingredients-list li:before {
    content: "• ";
    color: #667eea;
    font-weight: bold;
    margin-right: 0.5rem;
}

/* Loading Spinner */
.loading {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(255,255,255,0.8);
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 3000;
}

.spinner {
    width: 50px;
    height: 50px;
    border: 3px solid #f3f3f3;
    border-top: 3px solid #667eea;
    border-radius: 50%;
```

```
    animation: spin 1s linear infinite;
}

/* Toast Notifications */
.toast-container {
    position: fixed;
    top: 100px;
    right: 20px;
    z-index: 3000;
}

.toast {
    background: #333;
    color: white;
    padding: 1rem 1.5rem;
    border-radius: 5px;
    margin-bottom: 0.5rem;
    animation: slideInRight 0.3s ease-out;
    max-width: 300px;
}

.toast.success {
    background: #4caf50;
}

.toast.error {
    background: #f44336;
}

.toast.warning {
    background: #ff9800;
}

/* Quick Search Suggestions */
.quick-search {
    max-width: 800px;
    margin: 3rem auto 0;
    animation: fadeInUp 1s ease-out 0.6s both;
}

.quick-search-section {
    margin-bottom: 2rem;
    text-align: center;
}

.quick-search-section h3 {
    font-size: 1.1rem;
    margin-bottom: 1rem;
    color: rgba(255, 255, 255, 0.9);
    font-weight: 500;
}

.quick-search-section h3 i {
    margin-right: 0.5rem;
```

```
}

.quick-search-buttons {
    display: flex;
    flex-wrap: wrap;
    gap: 0.5rem;
    justify-content: center;
}

.quick-btn {
    background: rgba(255, 255, 255, 0.2);
    color: white;
    border: 1px solid rgba(255, 255, 255, 0.3);
    padding: 0.5rem 1rem;
    border-radius: 20px;
    cursor: pointer;
    font-size: 0.9rem;
    transition: all 0.3s;
    backdrop-filter: blur(10px);
}

.quick-btn:hover {
    background: rgba(255, 255, 255, 0.3);
    transform: translateY(-2px);
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
}

.quick-btn:active {
    transform: translateY(0);
}

/* Animations */
@keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
}

@keyframes fadeInUp {
    from {
        opacity: 0;
        transform: translateY(30px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

@keyframes slideIn {
    from {
        opacity: 0;
        transform: translateY(-50px);
    }
    to {
```

```
            opacity: 1;
            transform: translateY(0);
        }
    }

    @keyframes slideInRight {
        from {
            opacity: 0;
            transform: translateX(100px);
        }
        to {
            opacity: 1;
            transform: translateX(0);
        }
    }

    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }

    /* Responsive Design */
    @media (max-width: 768px) {
        body {
            padding-top: 70px; /* Adjust for smaller navbar on mobile */
        }

        .nav-menu {
            gap: 1rem;
        }

        .nav-brand {
            font-size: 1.2rem;
        }

        .hero h1 {
            font-size: 2rem;
        }

        .hero p {
            font-size: 1rem;
        }

        .search-container {
            flex-direction: column;
            gap: 1rem;
        }

        .recipe-grid {
            grid-template-columns: 1fr;
        }

        .modal-content {
            margin: 10% auto;
```

```
            padding: 1rem;
            max-width: 90%;
        }

        .form-row {
            grid-template-columns: 1fr;
        }

        .recipe-detail-meta {
            flex-direction: column;
            gap: 0.5rem;
        }

        .section-header {
            flex-direction: column;
            gap: 1rem;
            text-align: center;
        }

        .quick-search {
            margin-top: 2rem;
        }

        .quick-search-section h3 {
            font-size: 1rem;
        }

        .quick-btn {
            font-size: 0.85rem;
            padding: 0.4rem 0.8rem;
        }
    }
```