src/api/routers/recipe.py

Handles the user-facing actions around recipes such as listing, basic search/filtering, opening a single recipe, and creating/updating/removing recipes.

```
# src/api/routers/recipe.py
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from src.api.controllers import recipe as controller
from src.api.dependencies.database import get_db
from src.api.schemas.recipe import RecipeCreate, RecipeUpdate, RecipeRead
from src.api.util.auth import get_current_active_user,
get_current_active_admin_user
router = APIRouter(prefix="/recipes", tags=["Recipes"])
@router.post("/", response_model=RecipeRead,
dependencies=[Depends(get_current_active_user)])
def create(request: RecipeCreate, db: Session = Depends(get_db)):
    return controller.create(db, request)
@router.get("/", response_model=list[RecipeRead])
def read_all(db: Session = Depends(get_db)):
    return controller.read_all(db)
@router.get("/recent/", response_model=list[RecipeRead])
def read_recent(limit: int = 10, db: Session = Depends(get_db)):
    return controller.read_recent(db, limit)
@router.get("/search/", response_model=list[RecipeRead])
def search(query: str, threshold: int = 60, db: Session = Depends(get_db)):
    return controller.search(db, query, threshold)
@router.get("/{recipe_id}", response_model=RecipeRead)
def read_one(recipe_id: int, db: Session = Depends(get_db)):
    return controller.read_one(db, recipe_id)
@router.put("/{recipe_id}", response_model=RecipeRead,
dependencies=[Depends(get_current_active_user)])
def update(recipe_id: int, request: RecipeUpdate, db: Session =
Depends(get_db)):
    return controller.update(db, recipe_id, request)
@router.delete("/{recipe_id}",
dependencies=[Depends(get_current_active_admin_user)])
def delete(recipe_id: int, db: Session = Depends(get_db)):
    return controller.delete(db, recipe_id)
```

src/api/models/recipe.py

Defines what a “recipe” is in the system using a title and description, the step-by-step instructions, serving size, and optional embedded videos. It also stores which ingredients the recipe uses and timestamps.

```python
# src/api/models/recipe.py
from sqlalchemy import Integer, Column, String, DateTime, func
from src.api.dependencies.database import Base
class Recipe(Base):
    __tablename__ = "recipes"
    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, unique=True, index=True, nullable=False)
    description = Column(String, nullable=True)
    instructions = Column(String, nullable=False)
    ingredient_id_list = Column(String, nullable=False)
    servings = Column(Integer, nullable=False)
    video_embed_url = Column(String, nullable=True)
    image_url = Column(String, nullable=True)
    created_at = Column(DateTime, default=func.now())
    updated_at = Column(DateTime, default=func.now(), onupdate=func.now())

# src/api/models/pantry_ingredient.py
from sqlalchemy import Column, Integer, ForeignKey, String, DateTime, func
from sqlalchemy.orm import relationship
from src.api.dependencies.database import Base
class PantryIngredient(Base):
    __tablename__ = "pantry_ingredients"
    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    ingredient_id = Column(Integer, ForeignKey("ingredients.id"),
nullable=False)
    quantity = Column(String, nullable=False)
    unit = Column(String, nullable=False)
    created_at = Column(DateTime, default=func.now())
    updated_at = Column(DateTime, default=func.now(), onupdate=func.now())
    user = relationship("src.api.models.user.User",
back_populates="pantry_ingredients")
    ingredient = relationship("src.api.models.ingredient.Ingredient",
back_populates="pantry_ingredients")
```

src/api/routers/pantry_ingredient.py

Allows users to view their pantry, add new items, edit amounts, or remove entries. It always acts on the current logged-in user. Keeping everyone's pantry private and separate.

```
# src/api/routers/pantry_ingredient.py
from typing import Optional
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from src.api.controllers import pantry_ingredient as controller
from src.api.dependencies.database import get_db
from src.api.schemas.pantry_ingredient import PantryIngredientCreate,
PantryIngredientUpdate, PantryIngredientRead
from src.api.schemas.user import User as UserSchema
from src.api.util.auth import get_current_active_user
router = APIRouter(prefix="/pantryingredient",tags=["Pantry Ingredient"],
dependencies=[Depends(get_current_active_user)])
@router.post("/", response_model=PantryIngredientRead)
def create(request: PantryIngredientCreate, db: Session = Depends(get_db)):
    return controller.create(db, request)
@router.get("/", response_model=list[PantryIngredientRead])
def read_all(user_id: Optional[int] = None, db: Session = Depends(get_db)):
    return controller.read_all(db, user_id=user_id)
@router.get("/pantry", response_model=list[PantryIngredientRead])
def read_my_pantry(current_user: UserSchema =
Depends(get_current_active_user), db: Session = Depends(get_db)):
    pantry_items = controller.read_by_user(db, current_user.username)
    result = []
    for item in pantry_items:
       result.append(PantryIngredientRead(
          id=item.id, user_id=item.user_id,
          ingredient_id=item.ingredient_id, quantity=item.quantity,
          unit=item.unit,
          ingredient_name=item.ingredient.name if item.ingredient else None
       ))
    return result
@router.get("/{pantry_ingredient_id}", response_model=PantryIngredientRead)
def read_one(pantry_ingredient_id: int, db: Session = Depends(get_db)):
    return controller.read_one(db, pantry_ingredient_id)
@router.put("/{pantry_ingredient_id}", response_model=PantryIngredientRead)
def update(pantry_ingredient_id: int, request: PantryIngredientUpdate, db:
Session = Depends(get_db)):
    return controller.update(db, pantry_ingredient_id, request)
@router.delete("/{pantry_ingredient_id}")
def delete(pantry_ingredient_id: int, db: Session = Depends(get_db)):
    return controller.delete(db, pantry_ingredient_id)
```

src/website/js/recipeDetail.js

This loads the chosen recipe, formats the instructions for reading, shows an image/video when available.

```
# src/website/js/recipeDetail.js
function getUrlParameter(name);
async function fetchIngredient(ingredientId);
function loadRecipeDetails();
async function displayRecipe(recipe);
function formatInstructions(instructions);
function displayError(message);
document.addEventListener('DOMContentLoaded', loadRecipeDetails);

# src/website/js/auth.js
const API_BASE_URL = 'http://localhost:8000';
function getAccessToken();
function setAccessToken(token);
function removeAccessToken();
function getAuthHeaders();
function isAuthenticated();
function requireAuth();
async function getCurrentUser();
async function getCurrentUserId();
async function login(username, password);
function logout(redirectUrl = 'index.html');
async function register(username, email, password);
window.API_BASE_URL = API_BASE_URL;
```

src/api/routers/auth.py

Covers the sign-in, account creation, and a user check. Successful sign-in issues a time-limited access token; protected features (like pantry changes) rely on that token to identify the user.

```python
# src/api/routers/auth.py
from datetime import timedelta
from fastapi import APIRouter, Depends, HTTPException
from fastapi.security import OAuth2PasswordRequestForm
from sqlalchemy.orm import Session
from starlette import status
from src.api.controllers import user as user_controller
from src.api.dependencies.database import get_db
from src.api.schemas.user import User, UserCreate, UserRead
from src.api.util.auth import create_access_token,
ACCESS_TOKEN_EXPIRE_MINUTES, get_current_active_user
router = APIRouter(prefix="/auth", tags=["Authentication"])
@router.post("/login", response_model=dict)
async def login_for_access_token(form_data: OAuth2PasswordRequestForm =
Depends(), db: Session = Depends(get_db)):
    user = user_controller.authenticate_user(db, form_data.username,
form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect” headers={"WWW-Authenticate": "Bearer"})
    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(data={"sub": user.username},
expires_delta=access_token_expires)
    return {"access_token": access_token, "token_type": "bearer"}
@router.post("/register", response_model=User)
async def register_user(request: UserCreate, db: Session = Depends(get_db)):
    db_user = user_controller.read_user_by_username(db,
username=request.username)
    if db_user:
        raise HTTPException(status_code=400, detail="Already registered")
    db_user = user_controller.read_user_by_email(db, email=request.email)
    if db_user:
        raise HTTPException(status_code=400, detail="Already registered")
    new_user = user_controller.create(db=db, request=request)
    return User(
        username=new_user.username,email=new_user.email,
        is_active=new_user.is_active,role=new_user.role)
@router.get("/me", response_model=UserRead)
async def get_current_user_info(current_user: User =
Depends(get_current_active_user), db: Session = Depends(get_db)):
    db_user = user_controller.read_user_by_username(db,
username=current_user.username)
    if not db_user:
        raise HTTPException(status_code=404, detail="User not found")
    return UserRead.model_validate(db_user)
```

src/api/models/user.py

Defines user accounts, including login fields, active status, creation time, and a role type to distinguish regular users from administrators or moderators.

```
# src/api/models/user.py
import enum
from sqlalchemy import Column, Integer, String, Boolean, DateTime, func, Enum
from sqlalchemy.orm import relationship
from src.api.dependencies.database import Base
class Role(enum.Enum):
    User = "user"
    Administrator = "admin"
    Moderator = "moderator"
class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True, autoincrement=True)
    username = Column(String, unique=True, index=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String, nullable=False)
    is_active = Column(Boolean, default=True, nullable=False)
    created_at = Column(DateTime, default=func.now())
    role = Column(Enum(Role), default=Role.User, nullable=False)
    pantry_ingredients =
relationship("src.api.models.pantry_ingredient.PantryIngredient",
back_populates="user")
```

src/website/js/pantry.js

Loads the user's pantry, supports quick search within the list, and wires up add/edit/delete.

```
# src/website/js/pantry.js
function showToast(message, type = 'info', duration = 3000);
function showModal(options);
async function fetchAllIngredients();
async function searchIngredients(query);
async function createIngredient(name);
async function fetchMyPantry();
async function addToPantry(ingredientId, quantity, unit, userId);
async function updatePantryIngredient(pantryId, updates);
async function deletePantryIngredient(pantryId);
function renderPantryTable(pantryItems);
function filterPantryTable(query);
function showIngredientSuggestions(ingredients, input);
function initializeAddIngredientForm();
async function handleAddIngredient(ingredientInput, quantityInput,
unitInput);
async function editPantryItem(pantryId);
async function deletePantryItem(pantryId);
async function loadPantry();
function initializePantryPage();
window.editPantryItem = editPantryItem;
window.deletePantryItem = deletePantryItem;
document.addEventListener('DOMContentLoaded', initializePantryPage);

# src/website/js/search.js
async function searchRecipes(query, resultsContainerId, redirectToRecipes =
false);
function displaySearchResults(recipes, containerId, query);
async function loadRecentRecipes(containerId, limit = 10);
function initializeSearch();
if (document.readyState === 'loading') {
    document.addEventListener('DOMContentLoaded', initializeSearch);
} else {
    initializeSearch();
}
```

src/website/js/login.js

Performs basic field checks, friendly error messages, and redirects when sign-in succeeds.

```
# src/website/js/login.js
document.addEventListener('DOMContentLoaded', () => {
    const loginForm = document.getElementById('loginForm');
    const emailInput = document.getElementById('email');
    const passwordInput = document.getElementById('password');
    const emailError = document.getElementById('emailError');
    const passwordError = document.getElementById('passwordError');
    if (isAuthenticated()) {window.location.href = 'index.html'; return;}
    loginForm.addEventListener('submit', async (e) => {
        e.preventDefault();
        emailError.textContent = '';passwordError.textContent = '';
        const email = emailInput.value.trim();
        const password = passwordInput.value;
        if (!email) {
            emailError.textContent = 'Please enter your email or username';
            emailInput.focus(); return; }
        if (!password) {
            passwordError.textContent = 'Please enter your password';
            passwordInput.focus(); return; }
        const submitButton =
loginForm.querySelector('button[type="submit"]');
        const originalButtonText = submitButton.textContent;
        submitButton.disabled = true;
        submitButton.textContent = 'Signing in...';
        try {
            await login(email, password);
            window.location.href = 'index.html';
        } catch (error) {
            let errorMessage = error.message || 'Login failed';
            try {
                const errorData = JSON.parse(errorMessage);
                if (errorData.detail && Array.isArray(errorData.detail)) {
                    errorData.detail.forEach(err => {
                        passwordError.textContent = err.msg ||
err.ctx?.reason || 'Invalid input';
                    });
                } else if (typeof errorData.detail === 'string') {
                    passwordError.textContent = errorData.detail;
                } else {
                    passwordError.textContent = 'Login failed';
                }
            } catch (parseError) {passwordError.textContent = errorMessage;}
            submitButton.disabled = false;
            submitButton.textContent = originalButtonText;
        }
    });
});
```

src/website/RecipeDetail.html

The HTML for a single recipe view. It loads the detailed script to fill in the actual recipe content. For inspection purposes.

```html
# src/website/RecipeDetail.html
<html lang="en">
<head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>
    <title>Details • What Can We Cook</title>
    <link rel="icon" type="image/png" href="image/favicon.png"/>
    <link rel="stylesheet" href="css/stylesheet.css"/>
    <script src="js/recipeDetail.js" defer></script>
</head>
<body>
<header>
    <div class="wrap bar">
        <a class="brand" href="index.html">
            <img src="image/logo.png" alt="Logo" width="28" height="28"/>
            <span>What Can We Cook</span>
        </a>
        <nav>
            <a href="index.html">Home</a>
            <a href="Recipes.html">Recipes</a>
            <a href="Browse.html">Browse</a>
            <a href="Favorites.html">Favorites</a>
            <a href="MyPantry.html">My Pantry</a>
            <a href="Community.html">Community</a>
            <a href="SubmitRecipe.html" class="btn">Submit</a>
            <a href="Signup.html">Sign Up</a>
            <a href="SubmitRecipe.html" class="btn">Submit</a>
        </nav>
    </div>
</header>
<main class="wrap">
    <section class="hero"></section>
    <section class="recipe-hero" id="recipe-photo-section"></section>
    <section class="grid"></section>
</main>
<footer>
    <div class="wrap">
        <p class="small">© 2025 What Can We Cook</p>
    </div>
</footer>
</body>
</html>
```

src/website/MyPantry.html

Template for the pantry interface. A small "add ingredient" form with name/quantity/unit fields and the main table listing everything the user has.

```
# src/website/MyPantry.html
<html lang="en">
<head>
    <title>MyPantry • What Can We Cook</title>
    <link rel="icon" type="image/png" href="image/favicon.png"/>
    <link rel="stylesheet" href="css/stylesheet.css"/>
</head>
<body>
<header>
    <div class="wrap bar">
        <a class="brand" href="index.html">
            <img src="image/logo.png" alt="Logo" width="28" height="28"/>
            <span>What Can We Cook</span>
        </a>
        <nav></nav>
    </div>
</header>
<main class="wrap">
    <section class="hero">
        <h1>My Pantry</h1>
        <p class="muted">Manage your ingredients and see what you have at
home.</p>
    </section>
    <section class="pantry-container">
        <div class="card">
            <h3>Add Ingredient</h3>
            <div class="form-row">
                <div class="form-group">
                    <label for="ingredientNameInput">Ingredient Name</label>
                    <input type="text" id="ingredientNameInput"
placeholder="Start typing to search..." autocomplete="off">
                </div>
                <div class="form-group">
                    <label for="quantityInput">Quantity</label>
                    <input type="text" id="quantityInput">
                </div>
                <div class="form-group">
                    <label for="unitInput">Unit</label>
                    <input type="text" id="unitInput">
                </div>
                <div class="form-group">
                    <label>&nbsp;</label>
                    <button id="addIngredientBtn" class="btn
primary">Add</button>
                </div>
            </div>
```

```html
        </div>
        <div class="card">
            <div>
                <h3>Your Pantry</h3>
                <input type="text" id="pantrySearchInput">
            </div>
            <div id="emptyPantryState" class="empty-state">
                <p>Your pantry is empty!</p>
                <p class="small muted">Add ingredients above to get
started.</p>
            </div>
            <table id="pantryTable" class="pantry-table">
                <thead>
                <tr>
                    <th>Ingredient</th>
                    <th>Quantity</th>
                    <th>Unit</th>
                    <th>Actions</th>
                </tr>
                </thead>
                <tbody id="pantryTableBody"></tbody>
            </table>
        </div>
    </section>
</main>
<footer>
    <div class="wrap">
        <p class="small">© 2025 What Can We Cook</p>
    </div>
</footer>
<script src="js/auth.js"></script>
<script src="js/pantry.js"></script>
</body>
</html>
```

src/website/Recipes.html

This pulls in the search script so the page can fetch and render results dynamically.

```html
# src/website/Recipes.html
<html lang="en">
<head>
    <title>Recipes • What Can We Cook</title>
    <link rel="icon" type="image/png" href="image/favicon.png"/>
    <link rel="stylesheet" href="css/stylesheet.css"/>
    <script src="js/search.js" defer></script>
</head>
<body>
<header>
    <div class="wrap bar">
        <a class="brand" href="index.html">
            <img src="image/logo.png" alt="Logo" width="28" height="28"/>
            <span>What Can We Cook</span>
        </a>
        <nav></nav>
    </div>
</header>
<main class="wrap">
    <section class="hero">
        <h1>Recipes</h1>
        <p class="muted">Search by recipe name, description, or ingredients
below.</p>
        <div class="search">
            <input type="text" placeholder="Search recipes or type an
ingredient"/>
            <button type="button">Search</button>
        </div>
    </section>
    <section>
        <div class="grid auto" id="searchResults"></div>
    </section>
</main>
<footer>
    <div class="wrap">
        <p class="small">© 2025 What Can We Cook</p>
    </div>
</footer>
</body>
</html>
```

src/website/Login.html

The sign-in screen with labeled fields and links to sign up if needed.

```html
# src/website/Login.html
<html lang="en">
<head>
    <title>Login • What Can We Cook</title>
    <link rel="icon" type="image/png" href="image/favicon.png"/>
    <link rel="stylesheet" href="css/stylesheet.css"/>
</head>
<body>
<header>
    <div class="wrap bar">
        <a class="brand" href="index.html">
            <img src="image/logo.png" alt="Logo" width="28" height="28"/>
            <span>What Can We Cook</span>
        </a>
        <nav></nav>
    </div>
</header>
<main class="wrap">
    <section class="auth">
        <div class="auth-card">
            <h1>Welcome back</h1>
            <p class="muted mb">Sign in.</p>
            <form id="loginForm" action="index.html" method="POST">
                <div class="input-group">
                    <label for="email">Email or Username</label>
                    <input type="text" id="email" name="email"/>
                    <small class="error" id="emailError"></small>
                </div>
                <div class="input-group">
                    <label for="password">Password</label>
                    <input type="password" id="password" name="password"/>
                    <small class="error" id="passwordError"></small>
                </div>
                <button type="submit" class="btn primary full">Sign
In</button>
                <p class="muted center mt">Don't have an account? <a
class="link" href="Signup.html">Sign up</a></p>
            </form>
        </div>
    </section>
</main>
<script src="js/auth.js"></script>
<script src="js/login.js"></script>
</body>
</html>
```