

app.py:

```
from flask import Flask, send_from_directory
```

```
from flask_cors import CORS
```

```
from flask_jwt_extended import JWTManager
```

```
from flask_wtf.csrf import CSRFProtect
```

```
import sys
```

```
import os
```

```
# 添加当前目录到 Python 路径
```

```
sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
```

```
sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
```

```
from models.models import db
```

```
from routes.routes import api
```

```
from routes.ai_routes import ai_api
```

```
import os
```

```
import logging
```

```
# 配置日志
```

```
logging.basicConfig(
```

```
    level=logging.INFO,
```

```
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
```

```
    handlers=[
```

```
        logging.FileHandler('app.log'),
```

```
        logging.StreamHandler()
```

```
    ]
```

```
)
```

```
logger = logging.getLogger(__name__)
```

```
app = Flask(__name__)
```

```
CORS(app) # 允许跨域请求
```

```
# 配置
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///maturity.db'
```

```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

```
app.config['JWT_SECRET_KEY'] = 'your-secret-key-change-this-in-production'
```

```
app.config['SECRET_KEY'] = 'your-secret-key-for-csrf-protection'
```

```
app.config['WTF_CSRF_ENABLED'] = False
```

```
# 初始化 JWT 管理器
```

```
jwt = JWTManager(app)
```

```
# 初始化 CSRF 保护
```

```
# csrf = CSRFProtect(app)
```

```
# 初始化数据库
```

```
db.init_app(app)
```

```
# 注册蓝图
```

```
app.register_blueprint(api, url_prefix='/api')
```

```
app.register_blueprint(ai_api, url_prefix='/api')
```

```
# 静态文件服务
```

```
@app.route('/')

def serve_index():

    from flask import send_file

    # 手动指定项目目录路径

    project_dir = r'C:\Users\Lenovo\Documents\trae_projects'

    index_path = os.path.join(project_dir, 'frontend', 'html', 'index.html')


    # 记录日志

    logger.info(f"Serving index page: {index_path}")


    try:

        if os.path.exists(index_path):

            return send_file(index_path)

        else:

            logger.error(f"Index file not found: {index_path}")

            return 'File not found', 404

    except Exception as e:

        logger.error(f"Error serving index page: {str(e)}")

        return 'Internal server error', 500


@app.route('/css/<path:path>')

def serve_css(path):

    from flask import send_file

    project_dir = r'C:\Users\Lenovo\Documents\trae_projects'

    css_path = os.path.join(project_dir, 'frontend', 'css', path)


    logger.info(f"Serving CSS file: {css_path}")
```

```
try:

    if os.path.exists(css_path):

        return send_file(css_path)

    else:

        logger.error(f"CSS file not found: {css_path}")

        return 'File not found', 404

except Exception as e:

    logger.error(f"Error serving CSS file: {str(e)}")

    return 'Internal server error', 500
```

```
@app.route('/js/<path:path>')
```

```
def serve_js(path):

    from flask import send_file

    project_dir = r'C:\Users\Lenovo\Documents\trae_projects'

    js_path = os.path.join(project_dir, 'frontend', 'js', path)
```

```
    logger.info(f"Serving JS file: {js_path}")
```

```
try:

    if os.path.exists(js_path):

        return send_file(js_path)

    else:

        logger.error(f"JS file not found: {js_path}")

        return 'File not found', 404

except Exception as e:

    logger.error(f"Error serving JS file: {str(e)}")
```

```
return 'Internal server error', 500
```

```
@app.route('/pages/<path:path>')
```

```
def serve_pages(path):
```

```
    from flask import send_file
```

```
    project_dir = r'C:\Users\Lenovo\Documents\trae_projects'
```

```
    page_path = os.path.join(project_dir, 'frontend', 'html', path)
```

```
    logger.info(f"Serving page: {page_path}")
```

```
    try:
```

```
        if os.path.exists(page_path):
```

```
            return send_file(page_path)
```

```
        else:
```

```
            logger.error(f"Page not found: {page_path}")
```

```
            return 'File not found', 404
```

```
    except Exception as e:
```

```
        logger.error(f"Error serving page: {str(e)}")
```

```
        return 'Internal server error', 500
```

```
@app.route('/<filename>')
```

```
def serve_root_file(filename):
```

```
    from flask import send_file
```

```
    project_dir = r'C:\Users\Lenovo\Documents\trae_projects'
```

```
    file_path = os.path.join(project_dir, filename)
```

```
    logger.info(f"Serving root file: {file_path}")
```

```
try:

    if os.path.exists(file_path):

        return send_file(file_path)

    else:

        logger.error(f"Root file not found: {file_path}")

        return 'File not found', 404

except Exception as e:

    logger.error(f"Error serving root file: {str(e)}")

    return 'Internal server error', 500


# 创建上传目录

if not os.path.exists('uploads'):

    os.makedirs('uploads')


# 创建数据库表

with app.app_context():

    db.create_all()


if __name__ == '__main__':

    # 先以 HTTP 模式启动

    print("正在以 HTTP 模式启动...")

    app.run(debug=True, port=5000)
```

auth.py

```
<!DOCTYPE html>
```

```
<html lang="zh-CN">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>登录 / 注册 - 成熟度检测系统</title>
```

```
    <link rel="stylesheet" href="../css/navbar.css">
```

```
    <link rel="stylesheet" href="../css/auth.css">
```

```
</head>
```

```
<body>
```

```
    <div class="particles" id="particles"></div>
```

```
    <div class="floating-fruits">
```

```
        <span class="fruit fruit-1"> </span>
```

```
        <span class="fruit fruit-2"> </span>
```

```
        <span class="fruit fruit-3"> </span>
```

```
        <span class="fruit fruit-4"> </span>
```

```
    </div>
```

```
    <nav class="navbar">
```

```
        <a href="/pages/welcome.html" class="nav-logo"> <span data-i18n="app_name">成熟度检测</span></a>
```

```
        <div class="nav-links">
```

```
            <a href="/pages/welcome.html" data-i18n="nav_home">首页</a>
```

```
            <a href="/pages/index.html" data-i18n="nav_detect">检测</a>
```

```
            <a href="/pages/result.html" data-i18n="nav_result">结果</a>
```

```
<a href="/pages/models.html" data-i18n="nav_models">模型库</a>
<a href="/pages/feedback.html" data-i18n="nav_feedback">反馈</a>
<a href="/pages/history.html" data-i18n="nav_history">历史</a>
<a href="/pages/profile.html" data-i18n="nav_profile">我的</a>
<a href="/pages/settings.html" data-i18n="nav_settings">设置</a>
    <a href="/pages/ai-assistant.html">    AI 助手</a>
<a href="/pages/about.html">关于</a>
    <a href="/pages/auth.html" class="active">登录</a>
</div>
</nav>

<div class="container">
    <a href="/pages/welcome.html" class="back-btn">← 返回首页</a>

    <div class="auth-wrapper">
        <div class="auth-tabs">
            <button class="tab-btn active" data-tab="login">登录</button>
            <button class="tab-btn" data-tab="register">注册</button>
            <button class="tab-btn" data-tab="forgot">忘记密码</button>
            <button class="tab-btn" data-tab="reset">重置密码</button>
            <div class="tab-indicator"></div>
        </div>

        <div class="auth-content">
            <div class="auth-panel login-panel active" id="loginPanel">
                <div class="panel-header">
                    <div class="logo">    </div>
```


<h1>欢迎回来</h1>

<p>登录您的账户继续使用</p>

</div>

<form class="auth-form" id="loginForm">

<div class="input-group">

<div class="input-icon"> </div>

<input type="text" id="loginEmail" placeholder="请输入邮箱或手机号" required>

</div>

<div class="input-group">

<div class="input-icon"> </div>

<input type="password" id="loginPassword" placeholder="请输入密码" required>

<button type="button" class="toggle-password"> </button>

</div>

<div class="form-options">

<label class="remember-me">

<input type="checkbox" id="remember">

记住我

</label>

忘记密码?

直接修

改密码

</div>

<button type="submit" class="submit-btn">

登 录

→

</button>

</form>

</div>

<div class="auth-panel register-panel" id="registerPanel">

<div class="panel-header">

<div class="logo"> </div>

<h1>创建账户</h1>

<p>加入我们，开启智能检测之旅</p>

</div>

<form class="auth-form" id="registerForm">

<div class="input-group">

<div class="input-icon"> </div>

<input type="text" id="username" placeholder="请输入用户名" required>

</div>

<div class="input-group">

```
        <div class="input-icon">    </div>

        <input type="email" id="registerEmail" placeholder="
请输入邮箱" required>

    </div>
```

```
    <div class="input-group">

        <div class="input-icon">    </div>

        <input type="password" id="registerPassword"
placeholder="请输入密码" required>

        <button type="button"
class="toggle-password">    </button>

    </div>
```

```
    <div class="input-group">

        <div class="input-icon">    </div>

        <input type="password" id="confirmPassword"
placeholder="请确认密码" required>

        <button type="button"
class="toggle-password">    </button>

    </div>
```

```
    <div class="input-group">

        <div class="input-icon">    </div>

        <select id="userRole" required>

            <option value="">请选择用户角色</option>

            <option value="farmer">农业工作者</option>

            <option value="consumer">消费者</option>

            <option value="business">商业人员</option>
```

```
        </select>

    </div>

    <div class="password-strength">

        <div class="strength-bar">

            <div class="strength-fill" id="strengthFill"></div>

        </div>

        <span class="strength-text" id="strengthText">密码强
度</span>

    </div>

    <div class="form-options">

        <label class="remember-me">

            <input type="checkbox" id="agree" required>

            <span class="checkmark"></span>

            我已阅读并同意 <a href="#">服务条款</a>

        </label>

    </div>

    <button type="submit" class="submit-btn">

        <span class="btn-text">注 册</span>

        <span class="btn-icon">→</span>

    </button>

</form>

</div>
```

```
<div class="auth-panel forgot-panel" id="forgotPanel">

  <div class="panel-header">

    <div class="logo">    </div>

    <h1>忘记密码</h1>

    <p>请输入您的邮箱，我们将发送重置链接</p>

  </div>

  <form class="auth-form" id="forgotForm">

    <div class="input-group">

      <div class="input-icon">    </div>

      <input type="email" id="forgotEmail" placeholder="请
输入邮箱" required>

    </div>

    <button type="submit" class="submit-btn">

      <span class="btn-text">发送重置链接</span>

      <span class="btn-icon">→</span>

    </button>

  </form>

</div>

<div class="auth-panel reset-panel" id="resetPanel">

  <div class="panel-header">

    <div class="logo">    </div>

    <h1>重置密码</h1>

    <p>请设置新密码</p>
```

</div>

<form class="auth-form" id="resetForm">

<input type="hidden" id="resetToken">

<div class="input-group">

<div class="input-icon"> </div>

<input type="password" id="newPassword"
placeholder="请输入新密码" required>

<button type="button"
class="toggle-password"> </button>

</div>

<div class="input-group">

<div class="input-icon"> </div>

<input type="password" id="confirmNewPassword"
placeholder="请确认新密码" required>

<button type="button"
class="toggle-password"> </button>

</div>

<div class="password-strength">

<div class="strength-bar">

<div class="strength-fill"
id="resetStrengthFill"></div>

</div>

密
码强度

</div>

<button type="submit" class="submit-btn">

重置密码

→

</button>

</form>

</div>

</div>

</div>

</div>

<script src="../js/i18n.js"></script>

<script type="module" src="../js/auth-page.js"></script>

</body>

</html>

detection.py

```
<!DOCTYPE html>

<html lang="zh-CN">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>水果成熟度检测</title>

  <style>

    body {

      font-family: Arial, sans-serif;

      margin: 0;

      padding: 20px;

      background: #1a1a2e;

      color: #fff;

    }

    .container {

      max-width: 1200px;

      margin: 0 auto;

    }

    h1 {

      text-align: center;

      color: #4CAF50;

    }

    .tab-container {

      display: flex;

      justify-content: center;

      margin-bottom: 20px;
```



```
        flex-wrap: wrap;
    }

    .tab-btn {
        padding: 10px 20px;
        margin: 0 5px;
        background: #2a2a3e;
        color: #fff;
        border: none;
        border-radius: 5px;
        cursor: pointer;
        transition: all 0.3s ease;
    }

    .tab-btn.active {
        background: #4CAF50;
    }

    .detection-section {
        display: none;
        margin-bottom: 20px;
    }

    .detection-section.active {
        display: block;
    }

    .mode-container {
        display: flex;
        margin-bottom: 20px;
        flex-wrap: wrap;
    }
```

```
.mode-btn {  
    padding: 10px 15px;  
    margin: 0 5px;  
    background: #2a2a3e;  
    color: #fff;  
    border: none;  
    border-radius: 5px;  
    cursor: pointer;  
    transition: all 0.3s ease;  
}  
  
.mode-btn.active {  
    background: #4CAF50;  
}  
  
.upload-area {  
    border: 2px dashed #4CAF50;  
    border-radius: 10px;  
    padding: 30px;  
    text-align: center;  
    margin: 20px 0;  
    cursor: pointer;  
    transition: all 0.3s ease;  
}  
  
.upload-area:hover {  
    background: rgba(76, 175, 80, 0.1);  
}  
  
.upload-area.dragover {  
    background: rgba(76, 175, 80, 0.2);  
}
```

```
}

.preview-section {

    margin: 20px 0;

}

.preview-image {

    max-width: 100%;

    max-height: 400px;

    border-radius: 10px;

    border: 2px solid #4CAF50;

}

.camera-section {

    position: relative;

    margin: 20px 0;

    width: fit-content;

}

.camera-video {

    position: absolute;

    top: 0;

    left: 0;

    width: 100%;

    max-width: 640px;

    border-radius: 10px;

}

.camera-canvas {

    position: relative;

    z-index: 10;

    width: 100%;
```

```
        max-width: 640px;

        border-radius: 10px;

        display: block;
    }

    .button-container {

        margin: 20px 0;
    }

    .btn {

        padding: 10px 20px;

        margin: 0 5px;

        background: #4CAF50;

        color: #fff;

        border: none;

        border-radius: 5px;

        cursor: pointer;

        transition: all 0.3s ease;
    }

    .btn:hover {

        background: #45a049;
    }

    .loading-overlay {

        display: none;

        position: fixed;

        top: 0;

        left: 0;

        width: 100%;

        height: 100%;
```

```
        background: rgba(0, 0, 0, 0.7);

        justify-content: center;

        align-items: center;

        z-index: 1000;
    }

    .loading-spinner {

        width: 50px;

        height: 50px;

        border: 5px solid #f3f3f3;

        border-top: 5px solid #4CAF50;

        border-radius: 50%;

        animation: spin 1s linear infinite;
    }

    @keyframes spin {

        0% { transform: rotate(0deg); }

        100% { transform: rotate(360deg); }
    }

    .result-section {

        margin: 20px 0;

        padding: 20px;

        background: rgba(255, 255, 255, 0.1);

        border-radius: 10px;
    }

    .result-section h3 {

        color: #4CAF50;
    }

    .progress-bar {
```

```
width: 100%;

height: 20px;

background: #2a2a3e;

border-radius: 10px;

overflow: hidden;

margin: 10px 0;
}

.progress-fill {

height: 100%;

background: #4CAF50;

border-radius: 10px;

transition: width 1s ease;
}

.real-time-result {

margin: 20px 0;

padding: 15px;

background: rgba(255, 255, 255, 0.1);

border-radius: 10px;
}

@media (max-width: 768px) {

.tab-container, .mode-container {

flex-direction: column;
}

.tab-btn, .mode-btn {

margin: 5px 0;

width: 100%;
}
```

```
    }
</style>

</head>

<body>

    <div class="container">

        <h1>水果成熟度检测</h1>

        <div class="tab-container">

            <button class="tab-btn active" data-tab="cucumber">    黄瓜</button>

            <button class="tab-btn" data-tab="tomato">    番茄</button>

            <button class="tab-btn" data-tab="apple">    苹果</button>

        </div>

        <!-- 黄瓜检测 -->

        <div id="cucumberSection" class="detection-section active">

            <div class="mode-container">

                <button class="mode-btn active" data-mode="cucumber-image">
图片</button>

                <button class="mode-btn" data-mode="cucumber-video">视频
</button>

                <button class="mode-btn" data-mode="cucumber-camera">摄像
头</button>

            </div>

            <!-- 图片模式 -->

            <div id="cucumberImageSection" class="image-section">

                <div id="cucumberUploadArea" class="upload-area">

                    <p>点击或拖拽上传图片</p>
```

```
        <input type="file" id="cucumberFileInput" style="display: none;"
accept="image/*">
```

```
    </div>
```

```
</div>
```

```
<!-- 视频模式 -->
```

```
<div id="cucumberVideoSection" class="video-section" style="display:
none;">
```

```
    <div id="cucumberVideoUploadArea" class="upload-area">
```

```
        <p>点击或拖拽上传视频</p>
```

```
        <input type="file" id="cucumberVideoInput" style="display:
none;" accept="video/*">
```

```
    </div>
```

```
</div>
```

```
<!-- 摄像头模式 -->
```

```
<div id="cucumberCameraSection" class="camera-section" style="display:
none;">
```

```
    <video id="cucumberCameraVideo" class="camera-video"
autoplay></video>
```

```
    <canvas id="cucumberCameraCanvas"
class="camera-canvas"></canvas>
```

```
    <div class="button-container">
```

```
        <button id="cucumberStartCameraBtn" class="btn">启动摄像头
</button>
```

```
        <button id="cucumberCaptureBtn" class="btn" style="display:
none;">拍照</button>
```

```
    </div>
```

```
<div id="cucumberResult" class="real-time-result" style="display:
```



```
none;"></div>

</div>

<!-- 预览区域 -->

<div id="cucumberPreviewSection" class="preview-section"
style="display: none;">

    <img id="cucumberPreviewImage" class="preview-image" src=""
alt="预览图片">

</div>

<!-- 分析按钮 -->

<div class="button-container">

    <button id="cucumberAnalyzeBtn" class="btn">开始检测</button>

</div>

<!-- 结果区域 -->

<div id="cucumberResultSection" class="result-section" style="display:
none;">

    <h3>检测结果</h3>

    <p>水果类型: <span id="cucumberDetectedItem"></span></p>

    <p>成熟度: <span id="cucumberMaturityPercent"></span></p>

    <div class="progress-bar">

        <div id="cucumberProgressFill" class="progress-fill"
style="width: 0%"></div>

    </div>

    <p>状态: <span id="cucumberMaturityStatus"></span></p>

    <p>建议: <span id="cucumberRecommendation"></span></p>

</div>
```

</div>

<!-- 番茄检测 -->

<div id="tomatoSection" class="detection-section">

<div class="mode-container">

<button class="mode-btn active" data-mode="tomato-image">图
片</button>

<button class="mode-btn" data-mode="tomato-video">视频
</button>

<button class="mode-btn" data-mode="tomato-camera">摄像头
</button>

</div>

<!-- 图片模式 -->

<div id="tomatoImageSection" class="image-section">

<div id="tomatoUploadArea" class="upload-area">

<p>点击或拖拽上传图片</p>

<input type="file" id="tomatoFileInput" style="display: none;"
accept="image/*">

</div>

</div>

<!-- 视频模式 -->

<div id="tomatoVideoSection" class="video-section" style="display:
none;">

<div id="tomatoVideoUploadArea" class="upload-area">

<p>点击或拖拽上传视频</p>

<input type="file" id="tomatoVideoInput" style="display: none;"

```
accept="video/*">

    </div>

</div>

<!-- 摄像头模式 -->

<div id="tomatoCameraSection" class="camera-section" style="display:
none;">

    <video id="tomatoCameraVideo" class="camera-video"
autoplay></video>

    <canvas id="tomatoCameraCanvas"
class="camera-canvas"></canvas>

    <div class="button-container">

        <button id="tomatoStartCameraBtn" class="btn"> 启动摄像头
</button>

        <button id="tomatoCaptureBtn" class="btn" style="display:
none;">拍照</button>

    </div>

    <div id="tomatoResult" class="real-time-result" style="display:
none;"></div>

</div>

<!-- 预览区域 -->

<div id="tomatoPreviewSection" class="preview-section" style="display:
none;">

    <img id="tomatoPreviewImage" class="preview-image" src="" alt="
预览图片">

</div>

<!-- 分析按钮 -->
```

```
<div class="button-container">

    <button id="tomatoAnalyzeBtn" class="btn">开始检测</button>

</div>


<!-- 结果区域 -->

<div id="tomatoResultSection" class="result-section" style="display:
none;">

    <h3>检测结果</h3>

    <p>水果类型: <span id="tomatoDetectedItem"></span></p>

    <p>成熟度: <span id="tomatoMaturityPercent"></span></p>

    <div class="progress-bar">

        <div id="tomatoProgressFill" class="progress-fill" style="width:
0%"></div>

    </div>

    <p>状态: <span id="tomatoMaturityStatus"></span></p>

    <p>建议: <span id="tomatoRecommendation"></span></p>

</div>

</div>


<!-- 苹果检测 -->

<div id="appleSection" class="detection-section">

    <div class="mode-container">

        <button class="mode-btn active" data-mode="apple-image">图片
</button>

        <button class="mode-btn" data-mode="apple-video">视频
</button>

        <button class="mode-btn" data-mode="apple-camera">摄像头
</button>
```

```
</div>
```

```
<!-- 图片模式 -->
```

```
<div id="appleImageSection" class="image-section">
```

```
  <div id="appleUploadArea" class="upload-area">
```

```
    <p>点击或拖拽上传图片</p>
```

```
    <input type="file" id="appleFileInput" style="display: none;"  
accept="image/*">
```

```
  </div>
```

```
</div>
```

```
<!-- 视频模式 -->
```

```
<div id="appleVideoSection" class="video-section" style="display:  
none;">
```

```
  <div id="appleVideoUploadArea" class="upload-area">
```

```
    <p>点击或拖拽上传视频</p>
```

```
    <input type="file" id="appleVideoInput" style="display: none;"  
accept="video/*">
```

```
  </div>
```

```
</div>
```

```
<!-- 摄像头模式 -->
```

```
<div id="appleCameraSection" class="camera-section" style="display:  
none;">
```

```
  <video id="appleCameraVideo" class="camera-video"  
autoplay></video>
```

```
  <canvas id="appleCameraCanvas"  
class="camera-canvas"></canvas>
```

```
<div class="button-container">

    <button id="appleStartCameraBtn" class="btn">启动摄像头
</button>

    <button id="appleCaptureBtn" class="btn" style="display:
none;">拍照</button>

</div>

<div id="appleResult" class="real-time-result" style="display:
none;"></div>

</div>

<!-- 预览区域 -->

<div id="applePreviewSection" class="preview-section" style="display:
none;">

    <img id="applePreviewImage" class="preview-image" src="" alt="预
览图片">

</div>

<!-- 分析按钮 -->

<div class="button-container">

    <button id="appleAnalyzeBtn" class="btn">开始检测</button>

</div>

<!-- 结果区域 -->

<div id="appleResultSection" class="result-section" style="display:
none;">

    <h3>检测结果</h3>

    <p>水果类型: <span id="appleDetectedItem"></span></p>

    <p>成熟度: <span id="appleMaturityPercent"></span></p>
```

```
        <div class="progress-bar">
            <div id="appleProgressFill" class="progress-fill" style="width:
0%"></div>
```

```
        </div>
```

```
        <p>状态: <span id="appleMaturityStatus"></span></p>
```

```
        <p>建议: <span id="appleRecommendation"></span></p>
```

```
    </div>
```

```
</div>
```

```
<!-- 加载遮罩 -->
```

```
<div id="cucumberLoadingOverlay" class="loading-overlay">
```

```
    <div class="loading-spinner"></div>
```

```
</div>
```

```
<div id="tomatoLoadingOverlay" class="loading-overlay">
```

```
    <div class="loading-spinner"></div>
```

```
</div>
```

```
<div id="appleLoadingOverlay" class="loading-overlay">
```

```
    <div class="loading-spinner"></div>
```

```
</div>
```

```
</div>
```

```
<script type="module">
```

```
    // 导入 CSRF 处理函数
```

```
    import { fetchWithCSRF } from './js/csrf.js';
```

```
    document.addEventListener('DOMContentLoaded', function() {
```

```
        const tabBtns = document.querySelectorAll('.tab-btn');
```

```
const detectionSections =  
document.querySelectorAll('.detection-section');
```

```
tabBtns.forEach(btn => {  
    // 添加点击事件  
    btn.addEventListener('click', () => {  
        tabBtns.forEach(b => b.classList.remove('active'));  
        btn.classList.add('active');  
  
        const tab = btn.dataset.tab;  
        detectionSections.forEach(section => {  
            section.classList.remove('active');  
            if (section.id === tab + 'Section') {  
                section.classList.add('active');  
            }  
        });  
  
        stopAllCameras();  
    });  
});
```

```
initDetection('cucumber');  
initDetection('tomato');  
initDetection('apple');  
});
```

```
function stopAllCameras() {
```



```

const cameras = ['cucumber', 'tomato', 'apple'];

cameras.forEach(prefix => {

    const video = document.getElementById(prefix + 'CameraVideo');

    if (video && video.srcObject) {

        const stream = video.srcObject;

        stream.getTracks().forEach(track => track.stop());

        video.srcObject = null;

    }

    const startBtn = document.getElementById(prefix +
'StartCameraBtn');

    const captureBtn = document.getElementById(prefix + 'CaptureBtn');

    if (startBtn) startBtn.style.display = 'block';

    if (captureBtn) captureBtn.style.display = 'none';

});
}

```

```

function initDetection(prefix) {

    const uploadArea = document.getElementById(prefix + 'UploadArea');

    const fileInput = document.getElementById(prefix + 'FileInput');

    const videoUploadArea = document.getElementById(prefix +
'VideoUploadArea');

    const videoInput = document.getElementById(prefix + 'VideoInput');

    const previewSection = document.getElementById(prefix +
'PreviewSection');

    const previewImage = document.getElementById(prefix +
'PreviewImage');

    const analyzeBtn = document.getElementById(prefix + 'AnalyzeBtn');

```

```
const resultSection = document.getElementById(prefix + 'ResultSection');

const loadingOverlay = document.getElementById(prefix +
'LoadingOverlay');

const modeBtns =
document.querySelectorAll(`[data-mode^="${prefix}-"]`);

const imageSection = document.getElementById(prefix + 'ImageSection');

const videoSection = document.getElementById(prefix + 'VideoSection');

const cameraSection = document.getElementById(prefix +
'CameraSection');

const cameraVideo = document.getElementById(prefix + 'CameraVideo');

const cameraCanvas = document.getElementById(prefix +
'CameraCanvas');

const startCameraBtn = document.getElementById(prefix +
'StartCameraBtn');

const captureBtn = document.getElementById(prefix + 'CaptureBtn');


// 确保 Canvas 与视频大小相同
if (cameraCanvas) {
    cameraCanvas.width = cameraVideo.videoWidth || 640;
    cameraCanvas.height = cameraVideo.videoHeight || 480;
}


let currentMode = prefix + '-image';

let currentImage = null;

let currentVideo = null;

let cameraStream = null;

modeBtns.forEach(btn => {
```

```

// 添加点击事件

btn.addEventListener('click', () => {

    modeBtns.forEach(b => b.classList.remove('active'));

    btn.classList.add('active');

    currentMode = btn.dataset.mode;

    switchMode(currentMode, prefix);

});

});

function switchMode(mode, prefix) {

    imageSection.style.display = 'none';

    videoSection.style.display = 'none';

    cameraSection.style.display = 'none';

    previewSection.style.display = 'none';

    resultSection.style.display = 'none';

    stopCamera();

}

// 停止视频检测

if (window[prefix + 'StopVideoDetection']) {

    window[prefix + 'StopVideoDetection']();

}

if (mode === prefix + '-image') {

    imageSection.style.display = 'block';

} else if (mode === prefix + '-video') {

    videoSection.style.display = 'block';

} else if (mode === prefix + '-camera') {

```

```
        cameraSection.style.display = 'block';
    }
}

let realTimeDetectionInterval;

async function startCamera() {
    console.log('开始启动摄像头...');
    try {
        console.log('获取摄像头权限...');
        // 移除 facingMode 限制，让浏览器自动选择合适的摄像头
        cameraStream = await navigator.mediaDevices.getUserMedia({
            video: true
        });
        console.log('摄像头权限获取成功');
        cameraVideo.srcObject = cameraStream;
        console.log('视频流设置成功');

        // 等待视频元数据加载，然后设置 Canvas 尺寸
        cameraVideo.onloadedmetadata = function() {
            console.log('视频元数据加载完成，尺寸:',
cameraVideo.videoWidth, 'x', cameraVideo.videoHeight);
            if (cameraCanvas) {
                cameraCanvas.width = cameraVideo.videoWidth;
                cameraCanvas.height = cameraVideo.videoHeight;
                console.log('Canvas 尺寸设置成功');
            }
        }
    }
}
```

```
};

startCameraBtn.style.display = 'none';
captureBtn.style.display = 'block';
console.log('按钮状态更新成功');

// 启动实时检测
startRealTimeDetection();
} catch (err) {
    console.error('摄像头错误:', err);
    alert('无法访问摄像头，请检查：\n1. 浏览器是否已授予摄像头权限\n2. 摄像头是否被其他应用占用\n3. 设备是否有摄像头');
}
}

function stopCamera() {
    if (cameraStream) {
        cameraStream.getTracks().forEach(track => track.stop());
        cameraStream = null;
        cameraVideo.srcObject = null;
        startCameraBtn.style.display = 'block';
        captureBtn.style.display = 'none';

        // 停止实时检测
        if (realTimeDetectionInterval) {
            clearInterval(realTimeDetectionInterval);
            realTimeDetectionInterval = null;
        }
    }
}
```

```
        console.log('实时检测已停止');
    }
}

function captureImage() {
    console.log('开始拍照...');
    try {
        console.log('创建画布...');

        const canvas = document.createElement('canvas');
        canvas.width = cameraVideo.videoWidth;
        canvas.height = cameraVideo.videoHeight;
        console.log('画布创建成功， 尺寸:', canvas.width, 'x',
canvas.height);

        const ctx = canvas.getContext('2d');
        ctx.drawImage(cameraVideo, 0, 0);
        console.log('图像绘制成功');
        currentImage = canvas.toDataURL('image/jpeg');
        console.log('图像转换为 DataURL 成功');
        previewImage.src = currentImage;
        console.log('预览图像设置成功');
        previewSection.style.display = 'block';
        resultSection.style.display = 'none';
        console.log('预览区域显示成功');
        stopCamera();
        console.log('摄像头停止成功');
    } catch (err) {
```

```
        console.error('拍照错误:', err);

        alert('拍照失败, 请重试! ');
    }
}
```

```
function startRealTimeDetection() {

    console.log('开始实时检测...');

    // 显示实时结果区域

    const resultDiv = document.getElementById(prefix + 'Result');
    if (resultDiv) {
        resultDiv.style.display = 'block';
    }

    // 每 1000 毫秒分析一次画面, 减少服务器负载
    realTimeDetectionInterval = setInterval(async () => {

        try {

            const canvas = document.createElement('canvas');
            canvas.width = cameraVideo.videoWidth;
            canvas.height = cameraVideo.videoHeight;

            const ctx = canvas.getContext('2d');
            ctx.drawImage(cameraVideo, 0, 0);

            const imageData = canvas.toDataURL('image/jpeg');

            // 分析图像

            await analyzelImageForRealTime(imageData, prefix);

        } catch (err) {
```

```
        console.error('实时检测错误:', err);
    }
}, 1000);

// 实时绘制视频画面和检测结果
function drawCameraFeed() {
    if (cameraVideo.paused || cameraVideo.ended) return;

    if (cameraCanvas && cameraVideo.videoWidth > 0) {
        const ctx = cameraCanvas.getContext('2d');
        // 确保 Canvas 尺寸与视频一致
        if (cameraCanvas.width !== cameraVideo.videoWidth ||
cameraCanvas.height !== cameraVideo.videoHeight) {
            cameraCanvas.width = cameraVideo.videoWidth;
            cameraCanvas.height = cameraVideo.videoHeight;
        }
        // 清除画布
        ctx.clearRect(0, 0, cameraCanvas.width,
cameraCanvas.height);
        // 绘制视频画面
        ctx.drawImage(cameraVideo, 0, 0, cameraCanvas.width,
cameraCanvas.height);
    }

    // 继续下一帧
    requestAnimationFrame(drawCameraFeed);
}
```



```

        // 开始绘制

        drawCameraFeed();
    }

    async function analyzeImageForRealTime(imageData, currentPrefix) {
        try {
            // 将 base64 图片转换为 Blob

            const base64ToBlob = (base64, contentType = 'image/jpeg') =>

{
            const byteCharacters = atob(base64.split(',')[1]);

            const byteNumbers = new Array(byteCharacters.length);

            for (let i = 0; i < byteCharacters.length; i++) {
                byteNumbers[i] = byteCharacters.charCodeAt(i);
            }

            const byteArray = new Uint8Array(byteNumbers);

            return new Blob([byteArray], { type: contentType });
        };

        const blob = base64ToBlob(imageData);

        const formData = new FormData();

        formData.append('image', blob, 'image.jpg');

        formData.append('fruit_type', currentPrefix);

        console.log('开始发送实时检测 API 请求到: /api/detect');

        const apiResponse = await fetch('/api/detect', {
            method: 'POST',

            body: formData

```

```

    });

    if (apiResponse.ok) {
        const data = await apiResponse.json();
        displayRealTimeResult(data, currentPrefix);
    }
} catch (error) {
    console.error('实时分析错误:', error);
}
}

```

```

function displayRealTimeResult(data, currentPrefix) {
    // 在摄像头画面上显示检测结果

    const result = {
        itemName: {
            'cucumber': '  黄瓜',
            'tomato': '  番茄',
            'apple': '  苹果'
        },
        [currentPrefix],
        maturityPercent: data.result.maturity_level * 25, // 1-4 级对应
0-100%

        status: getMaturityStatus(data.result.maturity_level),
        recommendation: data.result.suggestion,
        maturityLevel: data.result.maturity_level
    };
}

```

```

// 显示实时结果

```

```
const resultDiv = document.getElementById(currentPrefix + 'Result');

if (resultDiv) {

    resultDiv.innerHTML = `

        <h3>${result.itemName}</h3>

        <p>成熟度: ${result.maturityPercent}%</p>

        <p>状态: ${result.status}</p>

        <p>建议: ${result.recommendation}</p>

    `;

    resultDiv.style.display = 'block';

}
```

```
// 更新实时结果显示
```

```
const maturityPercentEl = document.getElementById(currentPrefix +
'MaturityPercent');

const statusEl = document.getElementById(currentPrefix +
'MaturityStatus');

const recommendationEl = document.getElementById(currentPrefix +
'Recommendation');

if (maturityPercentEl) {

    maturityPercentEl.textContent = result.maturityPercent + '%';

}

if (statusEl) {

    statusEl.textContent = result.status;

}

if (recommendationEl) {

    recommendationEl.textContent = result.recommendation;

}
```

```
// 在摄像头画面上绘制检测结果

if (cameraCanvas) {

    const ctx = cameraCanvas.getContext('2d');

    // 清除画布

    ctx.clearRect(0, 0, cameraCanvas.width, cameraCanvas.height);

    // 绘制视频画面

    ctx.drawImage(cameraVideo, 0, 0, cameraCanvas.width,
cameraCanvas.height);

    // 绘制检测框和结果

    drawDetectionBox(ctx, result);

}

// 在视频画面上绘制检测结果

const videoCanvas = document.getElementById(currentPrefix +
'VideoCanvas');

if (videoCanvas) {

    const ctx = videoCanvas.getContext('2d');

    // 清除画布

    ctx.clearRect(0, 0, videoCanvas.width, videoCanvas.height);

    // 绘制检测框和结果

    drawDetectionBox(ctx, result);

}
```

```
}
```

```
function drawDetectionBox(ctx, result) {  
    // 获取 canvas 尺寸  
    const canvasWidth = ctx.canvas.width;  
    const canvasHeight = ctx.canvas.height;  
  
    // 模拟检测框位置（实际应该从 API 返回边界框数据）  
    // 这里假设水果在画面中央  
    const boxWidth = canvasWidth * 0.6;  
    const boxHeight = canvasHeight * 0.6;  
    const boxX = (canvasWidth - boxWidth) / 2;  
    const boxY = (canvasHeight - boxHeight) / 2;  
  
    // 根据成熟度等级选择边框颜色  
    let boxColor;  
    switch (result.maturityLevel) {  
        case 1: boxColor = '#FF4444'; break; // 未成熟 - 红色  
        case 2: boxColor = '#FFAA00'; break; // 半成熟 - 橙色  
        case 3: boxColor = '#44FF44'; break; // 成熟 - 绿色  
        case 4: boxColor = '#FF00FF'; break; // 过熟 - 紫色  
        default: boxColor = '#FFFFFF';  
    }  
  
    // 绘制边框  
    ctx.strokeStyle = boxColor;  
    ctx.lineWidth = 4;
```

```
ctx.strokeRect(boxX, boxY, boxWidth, boxHeight);
```

```
// 绘制成熟度等级标签
```

```
ctx.fillStyle = boxColor;
```

```
ctx.font = 'bold 24px Arial';
```

```
ctx.textAlign = 'center';
```

```
ctx.textBaseline = 'top';
```

```
// 计算文本位置
```

```
const textX = boxX + boxWidth / 2;
```

```
const textY = boxY - 30;
```

```
// 绘制背景
```

```
const text = `成熟度: ${result.status} (${result.maturityLevel}级)`;
```

```
const textWidth = ctx.measureText(text).width;
```

```
ctx.fillRect(textX - textWidth/2 - 10, textY - 5, textWidth + 20, 30);
```

```
// 绘制文本
```

```
ctx.fillStyle = '#000000';
```

```
ctx.fillText(text, textX, textY);
```

```
// 绘制建议文本
```

```
ctx.font = '16px Arial';
```

```
ctx.fillStyle = '#FFFFFF';
```

```
ctx.textAlign = 'left';
```

```
ctx.textBaseline = 'bottom';
```

```
// 绘制建议背景

const suggestion = result.recommendation;

const suggestionWidth = ctx.measureText(suggestion).width;

ctx.fillStyle = 'rgba(0, 0, 0, 0.7)';

ctx.fillRect(10, cameraCanvas.height - 50, suggestionWidth + 20, 40);


// 绘制建议文本

ctx.fillStyle = 'FFFFFF';

ctx.fillText(suggestion, 20, cameraCanvas.height - 20);
}


uploadArea.addEventListener('click', () => fileInput.click());


uploadArea.addEventListener('dragover', (e) => {

    e.preventDefault();

    uploadArea.classList.add('dragover');

});


uploadArea.addEventListener('dragleave', () => {

    uploadArea.classList.remove('dragover');

});


uploadArea.addEventListener('drop', (e) => {

    e.preventDefault();

    uploadArea.classList.remove('dragover');

    const files = e.dataTransfer.files;

    if (files.length > 0) {
```

```
        handleImageFile(files[0], prefix);
    }
});
```

```
fileInput.addEventListener('change', (e) => {
    console.log(`${prefix} fileInput change event triggered`);
    console.log(`${prefix} files length:`, e.target.files.length);
    if (e.target.files.length > 0) {
        console.log(`${prefix} file selected:`, e.target.files[0].name,
e.target.files[0].type);
        handleImageFile(e.target.files[0], prefix);
    } else {
        console.log(`${prefix} no file selected`);
    }
});
```

```
videoUploadArea.addEventListener('click', () => videoInput.click());
```

```
videoUploadArea.addEventListener('dragover', (e) => {
    e.preventDefault();
    videoUploadArea.classList.add('dragover');
});
```

```
videoUploadArea.addEventListener('dragleave', () => {
    videoUploadArea.classList.remove('dragover');
});
```



```
videoUploadArea.addEventListener('drop', (e) => {  
    e.preventDefault();  
    videoUploadArea.classList.remove('dragover');  
    const files = e.dataTransfer.files;  
    if (files.length > 0) {  
        handleVideoFile(files[0], prefix);  
    }  
});
```

```
videoInput.addEventListener('change', (e) => {  
    if (e.target.files.length > 0) {  
        handleVideoFile(e.target.files[0], prefix);  
    }  
});
```

```
startCameraBtn.addEventListener('click', startCamera);
```

```
captureBtn.addEventListener('click', captureImage);
```

```
function handleImageFile(file, prefix) {  
    if (!file.type.startsWith('image/')) {  
        alert('请上传图片文件! ');  
        return;  
    }  
}
```

```
console.log(`${prefix} image file selected:`, file.name, file.type);
```

```

const reader = new FileReader();

reader.onload = (e) => {

    console.log(`${prefix} image loaded successfully`);

    if (e.target.result) {

        currentImage = e.target.result;

        console.log(`${prefix} currentImage set:`,
currentImage.substring(0, 50) + '...');

        previewImage.src = currentImage;

        previewSection.style.display = 'block';

        resultSection.style.display = 'none';

        console.log(`${prefix} preview section displayed`);

    } else {

        alert('图片读取失败，请重试！');

        console.error(`${prefix} image read failed: e.target.result is
null`);

    }

};

reader.onerror = (e) => {

    alert('图片读取失败，请重试！');

    console.error(`${prefix} file reader error:`, e);

};

reader.readAsDataURL(file);

console.log(`${prefix} file reader started`);

}

```

```

function handleVideoFile(file, prefix) {

    if (!file.type.startsWith('video/')) {

```

```
    alert('请上传视频文件! ');  
    return;  
}
```

```
const reader = new FileReader();  
reader.onload = (e) => {  
    currentVideo = e.target.result;  
  
    // 创建视频元素用于播放  
    const videoElement = document.createElement('video');  
    videoElement.src = currentVideo;  
    videoElement.autoplay = true;  
    videoElement.loop = true;  
    videoElement.muted = true;  
    videoElement.style.width = '100%';  
    videoElement.style.maxWidth = '640px';  
  
    // 创建 Canvas 用于显示检测结果  
    const canvas = document.createElement('canvas');  
    canvas.id = prefix + 'VideoCanvas';  
    canvas.style.width = '100%';  
    canvas.style.maxWidth = '640px';  
    canvas.style.position = 'absolute';  
    canvas.style.top = '0';  
    canvas.style.left = '0';  
    canvas.style.zIndex = '10';
```

```
// 创建容器来存放视频和 Canvas

const videoContainer = document.createElement('div');

videoContainer.style.position = 'relative';

videoContainer.style.display = 'inline-block';

videoContainer.style.width = '100%';

videoContainer.style.maxWidth = '640px';


// 替换预览图片为视频和 Canvas

previewImage.style.display = 'none';

previewSection.innerHTML = '';

videoContainer.appendChild(videoElement);

videoContainer.appendChild(canvas);

previewSection.appendChild(videoContainer);

previewSection.style.display = 'block';

resultSection.style.display = 'none';


videoElement.onloadedmetadata = () => {

    console.log('视频元数据加载完成:',
videoElement.videoWidth, 'x', videoElement.videoHeight);

    canvas.width = videoElement.videoWidth;

    canvas.height = videoElement.videoHeight;


// 提取第一帧作为 currentImage, 以便用户点击"开始分析"
按钮时能够正常工作

const tempCanvas = document.createElement('canvas');

tempCanvas.width = videoElement.videoWidth;

tempCanvas.height = videoElement.videoHeight;
```

```

        const tempCtx = tempCanvas.getContext('2d');

        tempCtx.drawImage(videoElement, 0, 0);

        currentImage = tempCanvas.toDataURL('image/jpeg');

        console.log('视频第一帧提取成功， 长度:',
currentImage.length);

        // 开始播放视频

        videoElement.play();

        // 启动实时检测

        startVideoRealTimeDetection(videoElement, canvas, prefix);

    };

};

reader.readAsDataURL(file);
}

function startVideoRealTimeDetection(videoElement, canvas, prefix) {

    const ctx = canvas.getContext('2d');

    // 每 1000 毫秒分析一次画面， 减少服务器负载

    const videoDetectionInterval = setInterval(async () => {

        try {

            // 绘制视频帧到 Canvas

            ctx.drawImage(videoElement, 0, 0, canvas.width,
canvas.height);

            // 提取图像数据

```

```
const imageData = canvas.toDataURL('image/jpeg');

// 分析图像

await analyzeImageForRealTime(imageData, prefix);

} catch (err) {

    console.error('视频实时检测错误:', err);

}

}, 1000);

// 当切换模式时停止检测

const stopVideoDetection = () => {

    clearInterval(videoDetectionInterval);

    videoElement.pause();

    videoElement.src = '';

};

// 保存停止函数以便后续调用

window[prefix + 'StopVideoDetection'] = stopVideoDetection;

}

analyzeBtn.addEventListener('click', function() {

    console.log(`${prefix} analyzeBtn clicked`);

    analyzeImage();

});

async function analyzeImage() {

    console.log(`${prefix} analyzeImage function called`);
```

```

        console.log(`${prefix} analyzeImage called`);

        console.log(`${prefix} currentImage value:`, currentImage ?
currentImage.substring(0, 50) + '...' : 'null');

        console.log(`${prefix} previewSection display:`,
previewSection.style.display);

        console.log(`${prefix} previewImage src:`, previewImage.src);


        if (!currentImage) {

            console.error(`${prefix} currentImage is null, showing alert`);

            alert('请先上传图片、视频或拍摄照片！ ');

            return;

        }

        console.log(`${prefix} currentImage is not null, proceeding with
analysis`);


        loadingOverlay.style.display = 'flex';


        try {

            // 将 base64 图片转换为 Blob

            const base64ToBlob = (base64, contentType = 'image/jpeg') =>

{

            const byteCharacters = atob(base64.split(',')[1]);

            const byteNumbers = new Array(byteCharacters.length);

            for (let i = 0; i < byteCharacters.length; i++) {

                byteNumbers[i] = byteCharacters.charCodeAt(i);

            }

            const byteArray = new Uint8Array(byteNumbers);

            return new Blob([byteArray], { type: contentType });

        }

```

```
};

const blob = base64ToBlob(currentImage);
const formData = new FormData();
formData.append('image', blob, 'image.jpg');
formData.append('fruit_type', prefix);

console.log('FormData created:', formData);
console.log('Prefix:', prefix);
console.log('Current image length:', currentImage.length);
console.log('Blob size:', blob.size);
console.log('Blob type:', blob.type);

// 测试 FormData 内容
let formDataEntries = [];
for (let pair of formData.entries()) {
    formDataEntries.push({ key: pair[0], value: pair[1] });
}
console.log('FormData entries:', formDataEntries);

try {
    console.log('开始发送 API 请求到:
http://127.0.0.1:5000/api/detect');

    console.log('FormData size:', formData.size || 'FormData 对
象没有 size 属性');

    // 检查网络连接
```



```
console.log('检查网络连接...');

const networkStatus = navigator.onLine;

console.log('网络状态:', networkStatus);


// 发送检测请求

console.log('开始发送检测请求...');

// 使用相对路径，确保请求能够正确发送

const apiResponse = await fetch('/api/detect', {

    method: 'POST',

    body: formData

});


console.log('API Response Status:', apiResponse.status);

console.log('API Response Headers:',

Object.fromEntries(apiResponse.headers));


// 检查响应是否成功

console.log('API Response OK:', apiResponse.ok);


try {

    const data = await apiResponse.json();

    console.log('API Response Data:', data);


    if (apiResponse.ok) {

        const result = {

            itemName: {

                'cucumber': '  黄瓜',
```

```

        'tomato': ' 番茄',
        'apple': ' 苹果'
    }[prefix],
    maturityPercent: data.result.maturity_level *
25, // 1-4 级对应 0-100%
    status:
getMaturityStatus(data.result.maturity_level),
    recommendation: data.result.suggestion
};
displayResult(result, prefix);
} else {
    console.error('API Error:', data);
    alert('API 错误: ' + (data.message || '检测失败, 请
稍后重试'));
}
} catch (jsonError) {
    console.error('JSON Parse Error:', jsonError);
    const text = await apiResponse.text();
    console.error('API Response Text:', text);
    alert('JSON 解析错误: ' + text);
}
} catch (networkError) {
    console.error('Network Error:', networkError);
    console.error('Network Error Details:', {
        message: networkError.message,
        stack: networkError.stack,
        url: 'http://127.0.0.1:5000/api/detect',
        name: networkError.name,

```

```
        cause: networkError.cause

    });

    alert('网络错误，请稍后重试:' + networkError.message + '('
+ networkError.name + ')');

    // 尝试直接访问 API 地址，测试网络连接

    console.log('尝试直接访问 API 地址...');

    window.open('http://127.0.0.1:5000/api/detect', '_blank');

}

} catch (error) {

    console.error('检测错误:', error);

    alert('网络错误，请稍后重试');

} finally {

    loadingOverlay.style.display = 'none';

}

}
```

```
function getMaturityStatus(level) {

    const statuses = {

        1: '未成熟',

        2: '半成熟',

        3: '成熟',

        4: '过熟'

    };

    return statuses[level] || '未知';

}
```

```
function displayResult(result, prefix) {
```

```
        document.getElementById(prefix + 'DetectedItem').textContent =
result.itemName;

        document.getElementById(prefix + 'MaturityStatus').textContent =
result.status;

        document.getElementById(prefix + 'Recommendation').textContent =
result.recommendation;

        document.getElementById(prefix + 'MaturityPercent').textContent =
result.maturityPercent + '%';
```

```
const progressFill = document.getElementById(prefix + 'ProgressFill');
progressFill.style.width = '0%';
```

```
resultSection.style.display = 'block';
```

```
setTimeout(() => {
    progressFill.style.width = result.maturityPercent + '%';
}, 100);
```

```
// 存储检测结果到 localStorage
```

```
const detectionResult = {
    fruitType: prefix,
    fruitName: result.itemName,
    maturityPercent: result.maturityPercent,
    status: result.status,
    recommendation: result.recommendation,
    timestamp: new Date().toLocaleString(),
    image: currentImage,
    // 添加用户身份信息
```

```
        userRole: getUserRole(),
        userName: getUserUserName()
    };

    localStorage.setItem('detectionResult',
JSON.stringify(detectionResult));

    resultSection.scrollToView({ behavior: 'smooth', block: 'nearest' });
}
}
```

// 获取用户身份

```
function getUserRole() {
    const userInfo = localStorage.getItem('user');
    if (userInfo) {
        const user = JSON.parse(userInfo);
        return user.roleName || '消费者';
    }
    return '消费者';
}
```

// 获取用户名

```
function getUserUserName() {
    const userInfo = localStorage.getItem('user');
    if (userInfo) {
        const user = JSON.parse(userInfo);
        return user.username || '用户';
    }
}
```

```
        return '用户';
    }
</script>
</body>
</html>
```

```
routes.py

from flask import Blueprint, request, jsonify, Response

from flask_jwt_extended import create_access_token, jwt_required, get_jwt_identity

from werkzeug.security import check_password_hash

from models.models import db, User, Detection, Dataset

import os

import logging

import secrets

import requests

from datetime import datetime, timedelta

from ultralytics import YOLO

from utils.privacy import data_desensitizer, encryption_tool, access_control, audit_logger


# 配置日志
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)

logger = logging.getLogger(__name__)

api = Blueprint('api', __name__)


# 注册接口
@api.route('/register', methods=['POST'])
def register():
    try:
```

```
data = request.get_json()

# 脱敏处理后记录日志

if data:

    sanitized_data = data.copy()

    if 'email' in sanitized_data:

        sanitized_data['email'] =
data_desensitizer.desensitize_email(sanitized_data['email'])

    if 'password' in sanitized_data:

        sanitized_data['password'] = '***'

    logger.info(f"Register request: {sanitized_data}")

else:

    logger.error("No data provided")

    return jsonify({'message': '请提供注册信息'}), 400

required_fields = ['username', 'email', 'password', 'role']

for field in required_fields:

    if field not in data:

        logger.error(f"Missing field: {field}")

        return jsonify({'message': f'请提供{field}'}), 400

if User.query.filter_by(email=data['email']).first():

    logger.warning(f"Email already exists:
{data_desensitizer.desensitize_email(data['email'])}")

    return jsonify({'message': '邮箱已存在'}), 400

# 对密码进行加密处理

encrypted_password = encryption_tool.encrypt_data(data['password'])
```



```
new_user = User(
    username=data['username'],
    email=data['email'],
    password=encrypted_password, # 存储加密后的密码
    role=data['role']
)

db.session.add(new_user)

db.session.commit()


# 记录审计日志

audit_logger.log_data_access('system', 'user_registration', 'new user
registration')

logger.info(f"User registered successfully:
{data_desensitizer.desensitize_email(data['email'])}")

return jsonify({'message': '注册成功'}), 201

except Exception as e:

    logger.error(f"Error in register: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '注册失败，请稍后重试'}), 500


# 登录接口

@api.route('/login', methods=['POST'])

def login():

    try:

        data = request.get_json()
```

```
# 脱敏处理后记录日志

if data:

    sanitized_data = data.copy()

    if 'email' in sanitized_data:

        sanitized_data['email'] =
data_desensitizer.desensitize_email(sanitized_data['email'])

    if 'password' in sanitized_data:

        sanitized_data['password'] = '***'

    logger.info(f"Login request: {sanitized_data}")

else:

    logger.error("No data provided")

    return jsonify({'message': '请提供登录信息'}), 400


required_fields = ['email', 'password']

for field in required_fields:

    if field not in data:

        logger.error(f"Missing field: {field}")

        return jsonify({'message': f'请提供{field}'}), 400


user = User.query.filter_by(email=data['email']).first()

if not user:

    logger.warning(f"Invalid login attempt for email:
{data_desensitizer.desensitize_email(data['email'])}")

    return jsonify({'message': '邮箱或密码错误'}), 401


# 验证加密后的密码

decrypted_password = encryption_tool.decrypt_data(user.password)
```

```

        if decrypted_password != data['password']:

            logger.warning(f"Invalid login attempt for email:
{data_desensitizer.desensitize_email(data['email'])}")

            return jsonify({'message': '邮箱或密码错误'}), 401

        # 使用字符串类型的身份，避免 JWT 令牌验证问题

        access_token = create_access_token(identity=str(user.id))

        # 记录审计日志

        audit_logger.log_access(user.id, 'login', 'user account')

        logger.info(f"User logged in successfully:
{data_desensitizer.desensitize_email(data['email'])}")

        return jsonify({'access_token': access_token, 'user': {'id': user.id, 'username':
user.username, 'role': user.role}})

    except Exception as e:

        logger.error(f"Error in login: {str(e)}")

        import traceback

        traceback.print_exc()

        return jsonify({'message': '登录失败，请稍后重试'}), 500

# 检测接口（使用实际模型）

@api.route('/detect', methods=['POST'])

@api.route('/api/detect', methods=['POST'])

# @jwt_required()

def detect():

    try:

        # 尝试从 JWT 获取用户 ID，如果没有则使用默认消费者角色

        try:

            user_id = get_jwt_identity()

```

```
# 获取用户角色

user = User.query.filter_by(id=user_id).first()

user_role = user.role if user else 'consumer'

except:

    # 没有 JWT 令牌, 使用默认消费者角色

    user_id = 1 # 临时使用固定用户 ID

    user_role = 'consumer'


# 脱敏处理后记录日志

logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)}, User Role: {user_role}")


# 检查用户权限

if not access_control.check_permission(user_role, 'read'):

    logger.warning(f"Unauthorized access attempt by user: {data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限访问此功能'}), 403


# 获取上传的图片或图片 URL

logger.info(f"Files: {len(request.files) > 0}")

logger.info(f"Form keys: {list(request.form.keys())}")

logger.info(f"JSON data: {request.get_json()}")


# 记录所有请求参数

logger.info(f"All request form data: {dict(request.form)}")


image_path = None
```

```
# 支持微信小程序的 detectType 参数和 Web 端的 fruit_type 参数

detect_type = request.form.get('detectType') or request.form.get('fruit_type',
'unknown') or request.json.get('fruit_type', 'unknown')

# 将数字类型的 detectType 映射到对应的水果类型

type_mapping = {

    '0': 'cucumber', # 黄瓜

    '1': 'tomato',    # 番茄

    '2': 'apple'      # 苹果

}

fruit_type = type_mapping.get(detect_type, detect_type)

logger.info(f"Detect type: {detect_type}, Fruit type: {fruit_type}")


# 处理图片上传

if 'image' in request.files:

    image = request.files['image']

    logger.info(f"Image filename: {image.filename}")

    logger.info(f"Image content type: {image.content_type}")


# 保存图片

upload_folder = 'uploads'

if not os.path.exists(upload_folder):

    os.makedirs(upload_folder)

    logger.info(f"Created upload folder: {upload_folder}")


# 使用哈希值作为文件名，保护原始文件名信息

filename =

f"{datetime.now().strftime('%Y%m%d%H%M%S')}-{data_desensitizer.hash_identifier(image.filename[:8]).jpg}"
```

```
image_path = os.path.join(upload_folder, filename)

image.save(image_path)

logger.info(f"Image saved to: {image_path}")

# 处理图片 URL

elif 'image_url' in request.form or (request.json and 'image_url' in request.json):

    image_url = request.form.get('image_url') or request.json.get('image_url')

    logger.info(f"Image URL: {image_url}")


# 下载图片

upload_folder = 'uploads'

if not os.path.exists(upload_folder):

    os.makedirs(upload_folder)

    logger.info(f"Created upload folder: {upload_folder}")


# 使用哈希值作为文件名

filename =

f"{datetime.now().strftime('%Y%m%d%H%M%S')}-{data_desensitizer.hash_identifier(image_u
rl)[:8]}.jpg"

image_path = os.path.join(upload_folder, filename)


# 下载图片

try:

    response = requests.get(image_url, timeout=10)

    response.raise_for_status()

    with open(image_path, 'wb') as f:

        f.write(response.content)

    logger.info(f"Image downloaded from URL and saved to:
{image_path}")
```

```

except Exception as e:

    logger.error(f"Failed to download image from URL: {str(e)}")

    return jsonify({'message': '下载图片失败, 请检查 URL 是否有效'}), 400

else:

    logger.error("No image or image_url provided")

    return jsonify({'message': '请上传图片或提供图片 URL'}), 400


# 加载模型并进行预测

current_dir = os.path.dirname(os.path.abspath(__file__))

model_path = os.path.join(current_dir,
'runs/detect/fruit_maturity_v2/weights/best.pt')

if os.path.exists(model_path):

    logger.info(f"Loading model: {model_path}")

    model = YOLO(model_path)

try:

    # 进行预测, 降低置信度阈值以检测更多对象

    results = model(image_path, verbose=False, conf=0.25) # 降低置
信度阈值到 0.25


    # 解析预测结果

    if results and len(results) > 0:

        result = results[0]

        boxes = result.boxes


    # 添加详细日志

    logger.info(f"Detection boxes count: {len(boxes)}")

```

```

if len(boxes) > 0:
    for i, box in enumerate(boxes):
        logger.info(f"Box {i}: class={int(box.cls)},
conf={float(box.conf):.3f}")

# 找到置信度最高的检测框

best_box = None

max_conf = 0

for box in boxes:
    conf = float(box.conf)
    if conf > max_conf:
        max_conf = conf
        best_box = box

# 如果没有找到检测框，返回错误

if best_box is None:
    logger.warning("No box found")
    return jsonify({'message': '检测失败：未检测到指定类型
的水果'}), 400

# 检查置信度是否足够高

if max_conf < 0.3: # 降低阈值到 0.3，提高检测灵敏度
    logger.warning(f"Low confidence: {max_conf}")
    return jsonify({'message': '检测失败：未检测到指定类型
的水果'}), 400

best_class = int(best_box.cls)

```



```
logger.info(f"Selected class ID: {best_class}, Confidence:  
{max_conf}")
```

```
logger.info(f"Detected class: {best_class}, model name:  
{model.names[best_class]}")
```

色分析

```
# 无论检测到什么类别, 都根据用户指定的水果类型进行颜色分析
```

```
# 这样即使模型检测错误, 也能根据颜色判断正确的成熟度
```

```
if fruit_type == 'cucumber':
```

```
    # 使用颜色分析来判断黄瓜成熟度
```

```
    import cv2
```

```
    import numpy as np
```

```
    img = cv2.imread(image_path)
```

```
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
    if best_box is not None:
```

```
        x1, y1, x2, y2 = map(int, best_box.xyxy[0].tolist())
```

```
        cucumber_region = img_rgb[y1:y2, x1:x2]
```

```
        if cucumber_region.size > 0:
```

```
            avg_color =
```

```
np.mean(cucumber_region.reshape(-1, 3), axis=0)
```

```
            avg_r, avg_g, avg_b = avg_color
```

```
            green_intensity = avg_g - avg_r
```

```
            max_color =
```

```
np.max(cucumber_region.reshape(-1, 3), axis=1).mean()
```

```
            min_color =
```

```

np.min(cucumber_region.reshape(-1, 3), axis=1).mean()

saturation = (max_color - min_color) /
(max_color + 1e-6) * 100

brightness = (avg_r + avg_g + avg_b) / 3

logger.info(f"Color analysis: R={avg_r:.1f},
G={avg_g:.1f}, B={avg_b:.1f}, green_intensity={green_intensity:.1f},
saturation={saturation:.1f}%, brightness={brightness:.1f}")

# 根据颜色判断黄瓜成熟度
if avg_g > avg_r + 30 and saturation > 25 and
brightness > 90:

    maturity_level = 3 # 成熟
elif avg_g > avg_r + 20 and brightness > 80:
    maturity_level = 2 # 半熟
elif brightness < 70:
    maturity_level = 1 # 未熟
else:
    maturity_level = 4 # 过熟

logger.info(f"Color-based maturity level:
{maturity_level}")

else:

    maturity_level = 1 # 默认未熟
else:
    maturity_level = 1 # 默认未熟
elif fruit_type == 'apple':

    # 使用颜色分析来判断苹果成熟度

```

```

import cv2

import numpy as np

img = cv2.imread(image_path)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

if best_box is not None:

    x1, y1, x2, y2 = map(int, best_box.xyxy[0].tolist())

    apple_region = img_rgb[y1:y2, x1:x2]

    if apple_region.size > 0:

        avg_color =
np.mean(apple_region.reshape(-1, 3), axis=0)

        avg_r, avg_g, avg_b = avg_color

        # 计算红色和绿色强度
        red_intensity = avg_r - avg_g

        # 计算颜色饱和度
        max_color = np.max(apple_region.reshape(-1,
3), axis=1).mean()

        min_color = np.min(apple_region.reshape(-1,
3), axis=1).mean()

        saturation = (max_color - min_color) /
(max_color + 1e-6) * 100

        # 计算亮度
        brightness = (avg_r + avg_g + avg_b) / 3

        logger.info(f"Apple color analysis:
R={avg_r:.1f}, G={avg_g:.1f}, B={avg_b:.1f}, red_intensity={red_intensity:.1f},

```

```
saturation={saturation:.1f}%, brightness={brightness:.1f}")
```

```
# 根据颜色判断苹果成熟度
```

```
if avg_r > avg_g + 30 and saturation > 25:
```

```
# 红色系苹果，成熟
```

```
maturity_level = 3
```

```
elif avg_r > avg_g + 10:
```

```
# 淡红色，半熟
```

```
maturity_level = 2
```

```
elif avg_g > avg_r + 10:
```

```
# 绿色系，未熟
```

```
maturity_level = 1
```

```
else:
```

```
# 其他情况，根据亮度判断
```

```
if brightness > 120:
```

```
maturity_level = 3 # 成熟
```

```
else:
```

```
maturity_level = 2 # 半熟
```

```
logger.info(f"Color-based apple maturity level:
```

```
{maturity_level}")
```

```
else:
```

```
maturity_level = 1 # 默认未熟
```

```
else:
```

```
maturity_level = 1 # 默认未熟
```

```
elif fruit_type == 'tomato':
```

```
# 使用颜色分析来判断番茄成熟度
```

```

import cv2

import numpy as np

img = cv2.imread(image_path)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

if best_box is not None:

    x1, y1, x2, y2 = map(int, best_box.xyxy[0].tolist())

    tomato_region = img_rgb[y1:y2, x1:x2]

    if tomato_region.size > 0:

        avg_color =
np.mean(tomato_region.reshape(-1, 3), axis=0)

        avg_r, avg_g, avg_b = avg_color

        # 计算红色强度

        red_intensity = avg_r - avg_g

        # 计算颜色饱和度

        max_color =
np.max(tomato_region.reshape(-1, 3), axis=1).mean()

        min_color =
np.min(tomato_region.reshape(-1, 3), axis=1).mean()

        saturation = (max_color - min_color) /
(max_color + 1e-6) * 100

        # 计算亮度

        brightness = (avg_r + avg_g + avg_b) / 3

        logger.info(f"Tomato color analysis:
R={avg_r:.1f}, G={avg_g:.1f}, B={avg_b:.1f}, red_intensity={red_intensity:.1f},

```

```

saturation={saturation:.1f}%, brightness={brightness:.1f}")

# 根据颜色判断番茄成熟度
if avg_r > avg_g + 40 and avg_r > avg_b + 30
and saturation > 30:

    # 深红色, 饱和度高 -> 成熟/过熟
    if brightness > 120:
        maturity_level = 3 # 成熟
    else:
        maturity_level = 4 # 过熟
elif avg_r > avg_g + 20 and avg_r > avg_b +
10:

    # 中等红色 -> 半熟
    maturity_level = 2
elif avg_g > avg_r or avg_b > avg_r:
    # 绿色或青色 -> 未熟
    maturity_level = 1
else:
    # 其他情况 -> 成熟
    maturity_level = 3

logger.info(f"Color-based tomato maturity
level: {maturity_level}")

else:

    maturity_level = 1 # 默认未熟

else:

    maturity_level = 1 # 默认未熟

else:

```

```
fruit_type = 'cucumber' # 默认

maturity_level = 2

logger.warning(f"Unexpected class ID: {best_class}")

confidence = round(max_conf, 2)

logger.info(f"Detection result: fruit_type={fruit_type},
class_id={best_class}, maturity={maturity_level}, confidence={confidence}")
```

计算糖度和保鲜期

根据成熟度等级映射糖度范围

```
def get_sugar_content(fruit_type, maturity_level):
```

```
    sugar_ranges = {
```

```
        'cucumber': {
```

```
            1: {'min': 1.0, 'max': 2.0},
```

```
            2: {'min': 2.0, 'max': 3.0},
```

```
            3: {'min': 3.0, 'max': 4.0},
```

```
            4: {'min': 4.0, 'max': 5.0}
```

```
        },
```

```
        'apple': {
```

```
            1: {'min': 8.0, 'max': 10.0},
```

```
            2: {'min': 10.0, 'max': 12.0},
```

```
            3: {'min': 12.0, 'max': 14.0},
```

```
            4: {'min': 14.0, 'max': 16.0}
```

```
        },
```

```
        'tomato': {
```

```
            1: {'min': 3.0, 'max': 4.0},
```

```
            2: {'min': 4.0, 'max': 5.0},
```

```
        3: {'min': 5.0, 'max': 6.0},
        4: {'min': 6.0, 'max': 7.0}
    }
}

return sugar_ranges.get(fruit_type,
sugar_ranges['apple']).get(maturity_level, {'min': 0, 'max': 0})
```

根据成熟度等级和水果类型计算保鲜期（天）

```
def get_shelf_life(fruit_type, maturity_level):
```

```
    shelf_life = {
```

```
        'cucumber': {
```

```
            1: 14,
```

```
            2: 10,
```

```
            3: 7,
```

```
            4: 3
```

```
        },
```

```
        'apple': {
```

```
            1: 60,
```

```
            2: 45,
```

```
            3: 30,
```

```
            4: 15
```

```
        },
```

```
        'tomato': {
```

```
            1: 14,
```

```
            2: 10,
```

```
            3: 7,
```

```
            4: 3
```



```

        }
    }

    return shelf_life.get(fruit_type,
shelf_life['apple']).get(maturity_level, 7)

    sugar_content = get_sugar_content(fruit_type,
maturity_level)

    shelf_life = get_shelf_life(fruit_type, maturity_level)

# 保存检测结果到数据库
new_detection = Detection(
    user_id=user_id,
    fruit_type=fruit_type,
    maturity_level=maturity_level,
    confidence=confidence,
    image_path=image_path
)

db.session.add(new_detection)

db.session.commit()

logger.info(f"Detection saved to database:
{new_detection.id}")

# 记录审计日志
audit_logger.log_data_access(user_id, 'detection', 'fruit
maturity detection')

# 构建响应数据，同时兼容 Web 端和微信小程序
response_data = {

```

```
        'fruit_type': fruit_type,
        'maturity_level': maturity_level,
        'confidence': confidence,
        'sugar_content': sugar_content,
        'shelf_life': shelf_life,
        'suggestion': get_suggestion(fruit_type, maturity_level,
user_role)
    }
```

```
# 为微信小程序添加特定格式
```

```
# 计算成熟度百分比
```

```
maturity_percentage = str(maturity_level * 25) + '%'
```

```
return jsonify({
    'success': True,
    'data': {
        'maturity': maturity_percentage,
        'suggestion': get_suggestion(fruit_type,
maturity_level, user_role),
        'sugarContent':
f"{sugar_content['min']}-{sugar_content['max']}%" if 'min' in sugar_content else None,
        'freshness': f"{shelf_life}天" if shelf_life else None
    },
    'result': response_data # 保持 Web 端兼容
})
else:
    # 未检测到目标，返回错误
    logger.warning("No objects detected")
```

```

        return jsonify({'success': False, 'message': '检测失败：未检测到指定类型的水果'}), 400

    else:

        # 预测失败，返回错误

        logger.warning("Prediction failed")

        return jsonify({'success': False, 'message': '检测失败：未检测到指定类型的水果'}), 400

    except Exception as e:

        # 模型预测失败，返回错误

        logger.error(f"Model prediction failed: {str(e)}")

        import traceback

        traceback.print_exc()

        return jsonify({'success': False, 'message': '检测失败：未检测到指定类型的水果'}), 400

    else:

        # 模型文件不存在，返回错误

        logger.error(f"Model file not found: {model_path}")

        return jsonify({'success': False, 'message': '检测失败：未检测到指定类型的水果'}), 400

    except Exception as e:

        logger.error(f"Error in detect: {str(e)}")

        import traceback

        traceback.print_exc()

        return jsonify({'success': False, 'message': '检测失败，请稍后重试'}), 500

# 获取历史记录

@api.route('/history', methods=['GET'])

def get_history():

```

```
try:

    # 尝试从 JWT 获取用户 ID, 如果没有则使用默认用户 ID

    try:

        user_id = get_jwt_identity()

        user_id_int = int(user_id)

    except:

        # 没有 JWT 令牌, 使用默认用户 ID

        user_id_int = 1

    # 脱敏处理后记录日志

    logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id_int)} accessing history")

    # 检查用户权限

    user = User.query.filter_by(id=user_id_int).first()

    user_role = user.role if user else 'consumer'

    if not access_control.check_permission(user_role, 'read'):

        logger.warning(f"Unauthorized access attempt by user: {data_desensitizer.hash_identifier(user_id_int)}")

        return jsonify({'message': '无权限访问此功能'}), 403

    # 使用整数类型的 user_id_int 进行查询

    detections =

Detection.query.filter_by(user_id=user_id_int).order_by(Detection.created_at.desc()).all()

    history = []

    for d in detections:
```

```

        history.append({
            'id': d.id,
            'fruit_type': d.fruit_type,
            'maturity_level': d.maturity_level,
            'confidence': d.confidence,
            'created_at': d.created_at.strftime('%Y-%m-%d %H:%M:%S')
        })

# 记录审计日志
audit_logger.log_data_access(user_id_int, 'history', 'access detection history')

return jsonify({'history': history})

except Exception as e:
    logger.error(f"Error in get_history: {str(e)}")
    import traceback
    traceback.print_exc()
    return jsonify({'message': '获取历史记录失败'}), 500

# 删除历史记录 API
@api.route('/history/<int:history_id>', methods=['DELETE'])
def delete_history(history_id):
    try:
        # 尝试从 JWT 获取用户 ID，如果没有则使用默认用户 ID
        try:
            user_id = get_jwt_identity()
            user_id_int = int(user_id)
        except:

```

```
# 没有 JWT 令牌, 使用默认用户 ID

user_id_int = 1


# 脱敏处理后记录日志

logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id_int)} deleting
history: {history_id}")


# 检查用户权限

user = User.query.filter_by(id=user_id_int).first()

user_role = user.role if user else 'consumer'


if not access_control.check_permission(user_role, 'delete'):

    logger.warning(f"Unauthorized delete attempt by user:
{data_desensitizer.hash_identifier(user_id_int)}")

    return jsonify({'message': '无权限执行此操作'}), 403


# 查找并删除历史记录

detection = Detection.query.filter_by(id=history_id, user_id=user_id_int).first()

if not detection:

    logger.warning(f"History not found: {history_id} for user:
{data_desensitizer.hash_identifier(user_id_int)}")

    return jsonify({'message': '历史记录不存在'}), 404


db.session.delete(detection)

db.session.commit()


# 记录审计日志

audit_logger.log_data_access(user_id_int, 'history', f'delete detection history:
```

```
{history_id})
```

```
    return jsonify({'message': '历史记录删除成功'}), 200
```

```
except Exception as e:
```

```
    logger.error(f"Error in delete_history: {str(e)}")
```

```
    import traceback
```

```
    traceback.print_exc()
```

```
    return jsonify({'message': '删除历史记录失败'}), 500
```

```
# 批量删除历史记录 API
```

```
@api.route('/history/batch-delete', methods=['POST'])
```

```
def batch_delete_history():
```

```
    try:
```

```
        # 尝试从 JWT 获取用户 ID，如果没有则使用默认用户 ID
```

```
    try:
```

```
        user_id = get_jwt_identity()
```

```
        user_id_int = int(user_id)
```

```
    except:
```

```
        # 没有 JWT 令牌，使用默认用户 ID
```

```
        user_id_int = 1
```

```
    data = request.get_json()
```

```
    if not data or 'ids' not in data:
```

```
        return jsonify({'message': '缺少必要参数'}), 400
```

```
    history_ids = data['ids']
```

```
    # 脱敏处理后记录日志
```

```
logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id_int)} batch  
deleting history: {history_ids}")
```

```
# 检查用户权限
```

```
user = User.query.filter_by(id=user_id_int).first()
```

```
user_role = user.role if user else 'consumer'
```

```
if not access_control.check_permission(user_role, 'delete'):
```

```
logger.warning(f"Unauthorized batch delete attempt by user:  
{data_desensitizer.hash_identifier(user_id_int)}")
```

```
return jsonify({'message': '无权限执行此操作'}), 403
```

```
# 批量删除历史记录
```

```
deleted_count =
```

```
Detection.query.filter_by(user_id=user_id_int).filter(Detection.id.in_(history_ids)).delete(sy  
nchronize_session=False)
```

```
db.session.commit()
```

```
# 记录审计日志
```

```
audit_logger.log_data_access(user_id_int, 'history', f'batch delete detection  
history: {deleted_count} records')
```

```
return jsonify({'message': f'成功删除 {deleted_count} 条历史记录'}), 200
```

```
except Exception as e:
```

```
logger.error(f"Error in batch_delete_history: {str(e)}")
```

```
import traceback
```

```
traceback.print_exc()
```

```
return jsonify({'message': '批量删除历史记录失败'}), 500
```



```
# 导出历史记录 API

@api.route('/history/export', methods=['GET'])

def export_history():

    try:

        # 尝试从 JWT 获取用户 ID，如果没有则使用默认用户 ID

        try:

            user_id = get_jwt_identity()

            user_id_int = int(user_id)

        except:

            # 没有 JWT 令牌，使用默认用户 ID

            user_id_int = 1

        # 脱敏处理后记录日志

        logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id_int)} exporting history")

        # 检查用户权限

        user = User.query.filter_by(id=user_id_int).first()

        user_role = user.role if user else 'consumer'

        if not access_control.check_permission(user_role, 'read'):

            logger.warning(f"Unauthorized export attempt by user: {data_desensitizer.hash_identifier(user_id_int)}")

            return jsonify({'message': '无权限执行此操作'}), 403

        # 获取历史记录
```

```
detections =  
Detection.query.filter_by(user_id=user_id_int).order_by(Detection.created_at.desc()).all()
```

```
# 构建 CSV 数据
```

```
import csv
```

```
import io
```

```
import datetime
```

```
output = io.StringIO()
```

```
writer = csv.writer(output)
```

```
writer.writerow(['ID', '水果类型', '成熟度等级', '置信度', '检测时间'])
```

```
fruit_names = {
```

```
    'cucumber': '黄瓜',
```

```
    'tomato': '番茄',
```

```
    'apple': '苹果'
```

```
}
```

```
for d in detections:
```

```
    writer.writerow([
```

```
        d.id,
```

```
        fruit_names.get(d.fruit_type, d.fruit_type),
```

```
        d.maturity_level,
```

```
        f"{d.confidence * 100:.1f}%",
```

```
        d.created_at.strftime('%Y-%m-%d %H:%M:%S')
```

```
    ])
```

```
output.seek(0)

# 记录审计日志
audit_logger.log_data_access(user_id_int, 'history', 'export detection history')

# 返回 CSV 文件
return Response(output.getvalue(), mimetype='text/csv', headers={
    'Content-Disposition': f'attachment;
filename=history_export_{datetime.datetime.now().strftime("%Y%m%d_%H%M%S")}.csv'
})

except Exception as e:
    logger.error(f"Error in export_history: {str(e)}")
    import traceback
    traceback.print_exc()
    return jsonify({'message': '导出历史记录失败'}), 500

# 数据集管理接口
@api.route('/dataset/upload', methods=['POST'])
@jwt_required()
def upload_dataset():
    user_id = get_jwt_identity()

    # 脱敏处理后记录日志
    logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} uploading
dataset")

    # 检查用户权限
```

```
user = User.query.filter_by(id=user_id).first()

user_role = user.role if user else 'consumer'

if not access_control.check_permission(user_role, 'write'):

    logger.warning(f"Unauthorized upload attempt by user:
{data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限上传数据集'}), 403

if 'image' not in request.files:

    return jsonify({'message': '请上传图片'}), 400

image = request.files['image']

fruit_type = request.form.get('fruit_type', 'unknown')

maturity_level = request.form.get('maturity_level', 1)

# 创建数据集目录

dataset_dir = 'dataset/raw_images'

if not os.path.exists(dataset_dir):

    os.makedirs(dataset_dir)

# 使用哈希值作为文件名，保护原始文件名信息

filename =

f"{datetime.now().strftime('%Y%m%d%H%M%S')}_{data_desensitizer.hash_identifier(image.filename)[:8]}.jpg"

image_path = os.path.join(dataset_dir, filename)

image.save(image_path)

# 保存到数据库
```

```
new_dataset = Dataset(
    filename=filename,
    fruit_type=fruit_type,
    maturity_level=int(maturity_level),
    image_path=image_path,
    user_id=user_id
)
db.session.add(new_dataset)
db.session.commit()

# 记录审计日志
audit_logger.log_data_access(user_id, 'dataset', 'upload dataset')

return jsonify({
    'message': '数据集上传成功',
    'dataset': {
        'id': new_dataset.id,
        'filename': filename,
        'fruit_type': fruit_type,
        'maturity_level': maturity_level
    }
}), 201
```

```
@api.route('/dataset/list', methods=['GET'])
@jwt_required()
def get_dataset_list():
    user_id = get_jwt_identity()
```

```
# 脱敏处理后记录日志

logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} accessing dataset list")


# 检查用户权限

user = User.query.filter_by(id=user_id).first()

user_role = user.role if user else 'consumer'


if not access_control.check_permission(user_role, 'read'):

    logger.warning(f"Unauthorized access attempt by user: {data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限访问数据集'}), 403


# 获取查询参数

fruit_type = request.args.get('fruit_type')

maturity_level = request.args.get('maturity_level')

is_labeled = request.args.get('is_labeled')


# 构建查询

query = Dataset.query.filter_by(user_id=user_id)


if fruit_type:

    query = query.filter_by(fruit_type=fruit_type)


if maturity_level:

    query = query.filter_by(maturity_level=int(maturity_level))


if is_labeled is not None:
```

```

        query = query.filter_by(is_labeled=is_labeled.lower() == 'true')

    datasets = query.order_by(Dataset.created_at.desc()).all()

    dataset_list = []

    for d in datasets:

        dataset_list.append({

            'id': d.id,

            'filename': d.filename,

            'fruit_type': d.fruit_type,

            'maturity_level': d.maturity_level,

            'image_path': d.image_path,

            'is_labeled': d.is_labeled,

            'created_at': d.created_at.strftime('%Y-%m-%d %H:%M:%S')

        })

    # 记录审计日志

    audit_logger.log_data_access(user_id, 'dataset', 'access dataset list')

    return jsonify({'datasets': dataset_list, 'total': len(dataset_list)})

@api.route('/dataset/<int:dataset_id>', methods=['GET'])
@jwt_required()
def get_dataset(dataset_id):

    user_id = get_jwt_identity()

    # 脱敏处理后记录日志

```

```
logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} accessing dataset: {dataset_id}")
```

```
# 检查用户权限
```

```
user = User.query.filter_by(id=user_id).first()
```

```
user_role = user.role if user else 'consumer'
```

```
if not access_control.check_permission(user_role, 'read'):
```

```
    logger.warning(f"Unauthorized access attempt by user: {data_desensitizer.hash_identifier(user_id)}")
```

```
    return jsonify({'message': '无权限访问数据集'}), 403
```

```
dataset = Dataset.query.filter_by(id=dataset_id, user_id=user_id).first()
```

```
if not dataset:
```

```
    return jsonify({'message': '数据集不存在'}), 404
```

```
# 记录审计日志
```

```
audit_logger.log_data_access(user_id, 'dataset', f'access dataset {dataset_id}')
```

```
return jsonify({
```

```
    'id': dataset.id,
```

```
    'filename': dataset.filename,
```

```
    'fruit_type': dataset.fruit_type,
```

```
    'maturity_level': dataset.maturity_level,
```

```
    'image_path': dataset.image_path,
```

```
    'label_path': dataset.label_path,
```



```
        'is_labeled': dataset.is_labeled,  
        'created_at': dataset.created_at.strftime('%Y-%m-%d %H:%M:%S')  
    })
```

```
@api.route('/dataset/<int:dataset_id>', methods=['DELETE'])
```

```
@jwt_required()
```

```
def delete_dataset(dataset_id):
```

```
    user_id = get_jwt_identity()
```

```
    # 脱敏处理后记录日志
```

```
    logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} deleting dataset:  
{dataset_id}")
```

```
    # 检查用户权限
```

```
    user = User.query.filter_by(id=user_id).first()
```

```
    user_role = user.role if user else 'consumer'
```

```
    if not access_control.check_permission(user_role, 'delete'):
```

```
        logger.warning(f"Unauthorized delete attempt by user:  
{data_desensitizer.hash_identifier(user_id)}")
```

```
        return jsonify({'message': '无权限删除数据集'}), 403
```

```
    dataset = Dataset.query.filter_by(id=dataset_id, user_id=user_id).first()
```

```
    if not dataset:
```

```
        return jsonify({'message': '数据集不存在'}), 404
```

```
# 删除图片文件

if os.path.exists(dataset.image_path):

    os.remove(dataset.image_path)


# 删除标签文件

if dataset.label_path and os.path.exists(dataset.label_path):

    os.remove(dataset.label_path)


# 删除数据库记录

db.session.delete(dataset)

db.session.commit()


# 记录审计日志

audit_logger.log_data_access(user_id, 'dataset', f'delete dataset {dataset_id}')


return jsonify({'message': '数据集删除成功'})


@api.route('/dataset/<int:dataset_id>/label', methods=['PUT'])
@jwt_required()
def label_dataset(dataset_id):

    user_id = get_jwt_identity()


# 脱敏处理后记录日志

logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} labeling dataset:
{dataset_id}")


# 检查用户权限
```

```
user = User.query.filter_by(id=user_id).first()

user_role = user.role if user else 'consumer'

if not access_control.check_permission(user_role, 'write'):

    logger.warning(f"Unauthorized label attempt by user:
{data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限更新数据集标签'}), 403

dataset = Dataset.query.filter_by(id=dataset_id, user_id=user_id).first()

if not dataset:

    return jsonify({'message': '数据集不存在'}), 404

data = request.get_json()

# 更新标签信息

if 'label_data' in data:

    # 保存标签文件

    labels_dir = 'dataset/raw_labels'

    if not os.path.exists(labels_dir):

        os.makedirs(labels_dir)

    label_filename = f"{os.path.splitext(dataset.filename)[0]}.txt"

    label_path = os.path.join(labels_dir, label_filename)

    with open(label_path, 'w') as f:

        f.write(data['label_data'])
```

```
dataset.label_path = label_path
```

```
dataset.is_labeled = True
```

```
if 'maturity_level' in data:
```

```
    dataset.maturity_level = int(data['maturity_level'])
```

```
db.session.commit()
```

```
# 记录审计日志
```

```
audit_logger.log_data_access(user_id, 'dataset', f'label dataset {dataset_id}')
```

```
return jsonify({'message': '标签更新成功'})
```

```
@api.route('/dataset/statistics', methods=['GET'])
```

```
@jwt_required()
```

```
def get_dataset_statistics():
```

```
    user_id = get_jwt_identity()
```

```
# 脱敏处理后记录日志
```

```
    logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} accessing dataset  
statistics")
```

```
# 检查用户权限
```

```
user = User.query.filter_by(id=user_id).first()
```

```
user_role = user.role if user else 'consumer'
```

```
if not access_control.check_permission(user_role, 'read'):

    logger.warning(f"Unauthorized access attempt by user:
{data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限访问数据集统计'}), 403


# 获取统计数据

total_datasets = Dataset.query.filter_by(user_id=user_id).count()

labeled_datasets = Dataset.query.filter_by(user_id=user_id, is_labeled=True).count()

unlabeled_datasets = total_datasets - labeled_datasets


# 按水果类型统计

fruit_stats = db.session.query(

    Dataset.fruit_type,

    db.func.count(Dataset.id)

).filter_by(user_id=user_id).group_by(Dataset.fruit_type).all()


fruit_counts = {fruit: count for fruit, count in fruit_stats}


# 按成熟度统计

maturity_stats = db.session.query(

    Dataset.maturity_level,

    db.func.count(Dataset.id)

).filter_by(user_id=user_id).group_by(Dataset.maturity_level).all()


maturity_counts = {str(level): count for level, count in maturity_stats}


# 记录审计日志
```

```
audit_logger.log_data_access(user_id, 'dataset', 'access dataset statistics')
```

```
return jsonify({  
    'total': total_datasets,  
    'labeled': labeled_datasets,  
    'unlabeled': unlabeled_datasets,  
    'by_fruit_type': fruit_counts,  
    'by_maturity_level': maturity_counts  
})
```

辅助函数：根据成熟度和用户角色返回建议

```
def get_suggestion(fruit_type, maturity_level, user_role):
```

```
    # 不同角色的建议
```

```
    role_suggestions = {
```

```
        'farmer': { # 农业工作者
```

```
            1: '适合采摘后长途运输',
```

```
            2: '适合本地销售',
```

```
            3: '适合立即销售',
```

```
            4: '不适合销售，建议用作其他用途'
```

```
        },
```

```
        'business': { # 商业人员
```

```
            1: '适合批量采购储存',
```

```
            2: '适合中等批量采购',
```

```
            3: '适合小批量采购',
```

```
            4: '不建议采购'
```

```
        },
```

```
        'consumer': { # 消费者
```

```
        1: '还需等待一段时间再食用',
        2: '可以食用， 但口感一般',
        3: '最佳食用时机',
        4: '不建议食用'
    }
}
```

```
# 获取对应角色的建议， 如果角色不存在则使用消费者的建议
suggestions = role_suggestions.get(user_role, role_suggestions['consumer'])
return suggestions.get(maturity_level, '未知')
```

```
# 忘记密码请求接口
```

```
@api.route('/forgot-password', methods=['POST'])
```

```
def forgot_password():
```

```
    try:
```

```
        data = request.get_json()
```

```
        if not data or 'email' not in data:
```

```
            logger.error("No email provided")
```

```
            return jsonify({'message': '请提供邮箱地址'}), 400
```

```
        email = data['email']
```

```
        # 脱敏处理后记录日志
```

```
        logger.info(f"Forgot password request for email: {data_desensitizer.desensitize_email(email)}")
```

```
        # 查找用户
```

```

user = User.query.filter_by(email=email).first()

if not user:

    logger.warning(f"Forgot password attempt for non-existent email:
{data_desensitizer.desensitize_email(email)}")

    return jsonify({'message': '邮箱不存在'}), 404


# 生成重置令牌

reset_token = secrets.token_urlsafe(32)

reset_token_expiry = datetime.utcnow() + timedelta(hours=1) # 1 小时过期


# 更新用户的重置令牌和过期时间

user.reset_token = reset_token

user.reset_token_expiry = reset_token_expiry

db.session.commit()


# 记录审计日志

audit_logger.log_access(user.id, 'forgot_password', 'password reset request')

logger.info(f"Reset token generated for user:
{data_desensitizer.desensitize_email(email)}")


# 这里应该发送邮件，包含重置链接

# 由于是本地开发，我们直接返回令牌

reset_link = f"http://127.0.0.1:5000/pages/auth.html?token={reset_token}"


return jsonify({

    'message': '重置链接已生成',

    'reset_link': reset_link,

```



```

        'reset_token': reset_token # 仅用于开发测试
    }, 200

except Exception as e:

    logger.error(f"Error in forgot_password: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500


# 验证重置令牌接口

@api.route('/reset-password/<token>', methods=['GET'])

def verify_reset_token(token):

    try:

        # 查找用户

        user = User.query.filter_by(reset_token=token).first()

        if not user:

            logger.warning(f"Invalid reset token: {token[:8]}...")

            return jsonify({'message': '无效的重置令牌'}), 404


        # 检查令牌是否过期

        if user.reset_token_expiry and user.reset_token_expiry < datetime.utcnow():

            logger.warning(f"Expired reset token for user: {user.id}")

            return jsonify({'message': '重置令牌已过期'}), 400


        logger.info(f"Reset token verified for user: {user.id}")

        return jsonify({

            'message': '令牌有效',

            'user_id': user.id,

```

```

        'email': data_desensitizer.desensitize_email(user.email)
    }), 200

except Exception as e:

    logger.error(f"Error in verify_reset_token: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500


# 重置密码接口

@api.route('/reset-password', methods=['POST'])

def reset_password():

    try:

        data = request.get_json()

        if not data or 'token' not in data or 'new_password' not in data:

            logger.error("Missing token or new password")

            return jsonify({'message': '缺少必要参数'}), 400

        token = data['token']

        new_password = data['new_password']

        # 查找用户

        user = User.query.filter_by(reset_token=token).first()

        if not user:

            logger.warning(f"Invalid reset token: {token}")

            return jsonify({'message': '无效的重置令牌'}), 404

```

```
# 检查令牌是否过期

if user.reset_token_expiry and user.reset_token_expiry < datetime.utcnow():

    logger.warning(f"Expired reset token: {token}")

    return jsonify({'message': '令牌已过期'}), 400


# 重置密码

encrypted_password = encryption_tool.encrypt_data(new_password)

user.password = encrypted_password

user.reset_token = None

user.reset_token_expiry = None

db.session.commit()


# 记录审计日志

audit_logger.log_access(user.id, 'reset_password', 'password reset')

logger.info(f"Password reset successful for user:
{data_desensitizer.desensitize_email(user.email)}")


return jsonify({'message': '密码重置成功'}), 200

except Exception as e:

    logger.error(f"Error in reset_password: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500


# 修改密码接口

@api.route('/change-password', methods=['POST'])

def change_password():
```

```
try:

    data = request.get_json()

    if not data or 'user_id' not in data or 'current_password' not in data or
'new_password' not in data:

        logger.error("Missing user_id, current_password, or new_password")

        return jsonify({'message': '缺少必要参数'}), 400

    user_id = data['user_id']

    current_password = data['current_password']

    new_password = data['new_password']

    # 查找用户

    user = User.query.get(user_id)

    if not user:

        logger.warning(f"User not found: {user_id}")

        return jsonify({'message': '用户不存在'}), 404

    # 验证当前密码

    decrypted_password = encryption_tool.decrypt_data(user.password)

    if decrypted_password != current_password:

        logger.warning(f"Incorrect current password for user: {user_id}")

        return jsonify({'message': '当前密码错误'}), 401

    # 更新密码

    encrypted_password = encryption_tool.encrypt_data(new_password)

    user.password = encrypted_password
```

```
db.session.commit()

# 记录审计日志

audit_logger.log_access(user_id, 'change_password', 'password changed')

logger.info(f"Password changed successful for user:
{data_desensitizer.desensitize_email(user.email)}")

return jsonify({'message': '密码修改成功'}), 200

except Exception as e:

    logger.error(f"Error in change_password: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500

# 直接修改密码接口（无需当前密码）

@api.route('/direct-change-password', methods=['POST'])

def direct_change_password():

    try:

        data = request.get_json()

        if not data or 'email' not in data or 'new_password' not in data:

            logger.error("Missing email or new_password")

            return jsonify({'message': '缺少必要参数'}), 400

        email = data['email']

        new_password = data['new_password']
```

```

# 查找用户

user = User.query.filter_by(email=email).first()

if not user:

    logger.warning(f"User not found for email:
{data_desensitizer.desensitize_email(email)}")

    return jsonify({'message': '邮箱不存在'}), 404


# 更新密码

encrypted_password = encryption_tool.encrypt_data(new_password)

user.password = encrypted_password

db.session.commit()


# 记录审计日志

audit_logger.log_access(user.id, 'direct_change_password', 'password changed
without current password')

logger.info(f"Direct password change successful for user:
{data_desensitizer.desensitize_email(user.email)}")


return jsonify({'message': '密码修改成功'}), 200

except Exception as e:

    logger.error(f"Error in direct_change_password: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500


# 获取 CSRF 令牌接口

@api.route('/csrf-token', methods=['GET'])

def get_csrf_token():

```

```
from flask_wtf.csrf import generate_csrf

return jsonify({'csrf_token': generate_csrf()})


# 获取用户信息接口

@api.route('/user-info', methods=['GET'])
@jwt_required()
def get_user_info():
    try:
        user_id = get_jwt_identity()

        # 查找用户
        user = User.query.get(user_id)

        if not user:
            logger.warning(f"User not found: {user_id}")
            return jsonify({'message': '用户不存在'}), 404

        # 构建用户信息
        user_info = {
            'id': user.id,
            'username': user.username,
            'email': user.email,
            'role': user.role,
            'roleName': {
                'farmer': '农业工作者',
                'consumer': '消费者',
                'business': '商业人员'
            }.get(user.role, user.role)
        }
```

```
}
```

```
    logger.info(f"User info retrieved for user:  
{data_desensitizer.desensitize_email(user.email)}")
```

```
    return jsonify({'user': user_info}), 200
```

```
except Exception as e:
```

```
    logger.error(f"Error in get_user_info: {str(e)}")
```

```
    import traceback
```

```
    traceback.print_exc()
```

```
    return jsonify({'message': '处理失败, 请稍后重试'}), 500
```

```
# 更新用户信息接口
```

```
@api.route('/update-user', methods=['PUT'])
```

```
@jwt_required()
```

```
def update_user():
```

```
    try:
```

```
        user_id = get_jwt_identity()
```

```
        data = request.get_json()
```

```
    # 查找用户
```

```
    user = User.query.get(user_id)
```

```
    if not user:
```

```
        logger.warning(f"User not found: {user_id}")
```

```
        return jsonify({'message': '用户不存在'}), 404
```

```
    # 更新用户信息
```

```
    if 'username' in data:
```



```
        user.username = data['username']

    db.session.commit()

    # 记录审计日志
    audit_logger.log_access(user_id, 'update_user', 'user information updated')

    logger.info(f"User info updated for user:
{data_desensitizer.desensitize_email(user.email)}")

    return jsonify({'message': '用户信息更新成功'}), 200
except Exception as e:
    logger.error(f"Error in update_user: {str(e)}")
    import traceback
    traceback.print_exc()
    return jsonify({'message': '处理失败，请稍后重试'}), 500
```

history.py

```
<!DOCTYPE html>
```

```
<html lang="zh-CN">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>历史记录 - 成熟度检测系统</title>
```

```
    <link rel="stylesheet" href="../css/navbar.css">
```

```
    <link rel="stylesheet" href="../css/history.css">
```

```
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

```
    <script src="/js/i18n.js"></script>
```

```
</head>
```

```
<body>
```

```
    <div class="particles" id="particles"></div>
```

```
    <nav class="navbar">
```

```
        <a href="/pages/welcome.html" class="nav-logo">                <span
data-i18n="app_name">成熟度检测</span></a>
```

```
        <div class="nav-links">
```

```
            <a href="/pages/welcome.html" data-i18n="nav_home">首页</a>
```

```
            <a href="/pages/index.html" data-i18n="nav_detect">检测</a>
```

```
            <a href="/pages/result.html" data-i18n="nav_result">结果</a>
```

```
            <a href="/pages/models.html" data-i18n="nav_models">模型库</a>
```

```
            <a href="/pages/feedback.html" data-i18n="nav_feedback">反馈</a>
```

```
            <a href="/pages/history.html" class="active" data-i18n="nav_history">历
史记录</a>
```

```
            <a href="/pages/profile.html" data-i18n="nav_profile">个人中心</a>
```

[AI 助手](/pages/ai-assistant.html)

[登录/注册](/pages/auth.html)

检测历史

查看您所有的检测记录

86

总检测次数

28

黄瓜检测

```
<span class="stat-icon"> </span>

<div class="stat-info">

    <span class="stat-value">32</span>

    <span class="stat-label">番茄检测</span>

</div>

</div>

<div class="stat-box">

    <span class="stat-icon"> </span>

    <div class="stat-info">

        <span class="stat-value">26</span>

        <span class="stat-label">苹果检测</span>

    </div>

</div>

</div>

<div class="charts-section">

    <h2> 检测数据分析</h2>

    <div class="charts-grid">

        <div class="chart-container">

            <h3>成熟度趋势</h3>

            <canvas id="maturityTrendChart"></canvas>

        </div>

        <div class="chart-container">

            <h3>水果类型分布</h3>

            <canvas id="fruitDistributionChart"></canvas>

        </div>

        <div class="chart-container">
```

<h3>成熟度等级分布</h3>

<canvas id="maturityLevelChart"></canvas>

</div>

<div class="chart-container">

<h3>检测频率</h3>

<canvas id="detectionFrequencyChart"></canvas>

</div>

</div>

</div>

<div class="filter-bar">

<div class="search-box">

<input type="text" data-i18n-placeholder="search_model" placeholder="搜索检测记录..." id="searchInput">

<button class="search-btn"> </button>

</div>

<div class="filter-options">

<select id="fruitFilter">

<option value="all">全部产品</option>

<option value="cucumber"> 黄瓜</option>

<option value="tomato"> 番茄</option>

<option value="apple"> 苹果</option>

</select>

<select id="timeFilter">

<option value="all">全部时间</option>

<option value="today">今天</option>

<option value="week">本周</option>

```
        <option value="month">本月</option>
    </select>

    <select id="sortFilter">

        <option value="newest">最新优先</option>

        <option value="oldest">最早优先</option>

        <option value="score">评分优先</option>

    </select>

</div>

</div>

<div class="history-groups history-container">

    <!-- 历史记录将通过 JavaScript 动态生成 -->

</div>

<div class="pagination">

    <button class="page-btn" disabled>上一页</button>

    <div class="page-numbers">

        <button class="page-number active">1</button>

        <button class="page-number">2</button>

        <button class="page-number">3</button>

        <span class="page-ellipsis">...</span>

        <button class="page-number">10</button>

    </div>

    <button class="page-btn">下一页</button>

</div>

<div class="batch-actions">
```

```
<label class="select-all">

    <input type="checkbox" id="selectAll">

    <span>全选</span>

</label>

<button class="batch-btn export-btn">    <span>导出
</span></button>

    <button class="batch-btn delete-batch-btn">    <span>批量删除
</span></button>

</div>

</div>

<script src="/js/i18n.js"></script>

<script src="/js/history.js"></script>

</body>

</html>
```

welcome.py

```
<!DOCTYPE html>
```

```
<html lang="zh-CN">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>欢迎 - 成熟度检测系统</title>
```

```
    <link rel="stylesheet" href="../css/navbar.css">
```

```
    <link rel="stylesheet" href="../css/welcome.css">
```

```
</head>
```

```
<body>
```

```
    <div class="particles" id="particles"></div>
```

```
    <nav class="navbar">
```

```
        <a href="/pages/welcome.html" class="nav-logo">          <span  
data-i18n="app_name">成熟度检测</span></a>
```

```
        <div class="nav-links">
```

```
            <a href="/pages/welcome.html" class="active" data-i18n="nav_home">  
首页</a>
```

```
            <a href="/pages/index.html" data-i18n="nav_detect">检测</a>
```

```
            <a href="/pages/result.html" data-i18n="nav_result">结果</a>
```

```
            <a href="/pages/models.html" data-i18n="nav_models">模型库</a>
```

```
            <a href="/pages/feedback.html" data-i18n="nav_feedback">反馈</a>
```

```
            <a href="/pages/history.html" data-i18n="nav_history">历史</a>
```

```
            <a href="/pages/profile.html" data-i18n="nav_profile">我的</a>
```

```
            <a href="/pages/settings.html" data-i18n="nav_settings">设置</a>
```

```
            <a href="/pages/ai-assistant.html">    AI 助手</a>
```



```
<a href="/pages/about.html" data-i18n="nav_about">关于</a>

      <a href="/pages/auth.html" class="login-btn" data-i18n="nav_login">登
录 / 注册</a>

    </div>

  </nav>

  <div class="floating-fruits">

    <span class="fruit fruit-1"> </span>

    <span class="fruit fruit-2"> </span>

    <span class="fruit fruit-3"> </span>

    <span class="fruit fruit-4"> </span>

    <span class="fruit fruit-5"> </span>

    <span class="fruit fruit-6"> </span>

    <span class="fruit fruit-7"> </span>

    <span class="fruit fruit-8"> </span>

  </div>

  <div class="container">

    <div class="hero">

      <div class="glitch-wrapper">

        <h1 class="glitch" data-text="欢迎来到" data-i18n="welcome_text">
欢迎来到</h1>

      </div>

      <div class="title-container">

        <h2 class="gradient-text" data-i18n="welcome_title">黄瓜 苹果 番
茄检测系统</h2>

        <div class="underline"></div>

      </div>

    </div>

  </div>
```

</div>

<div class="typewriter-container">

<p class="typewriter" id="typewriter"></p>

|

</div>

<div class="features">

<div class="feature-card" data-delay="0">

<div class="feature-icon"> </div>

<h3 data-i18n="feature_1_title">智能识别</h3>

<p data-i18n="feature_1_desc">上传图片即可检测</p>

</div>

<div class="feature-card" data-delay="1">

<div class="feature-icon"> </div>

<h3 data-i18n="feature_2_title">精准分析</h3>

<p data-i18n="feature_2_desc">AI 驱动的成熟度判断</p>

</div>

<div class="feature-card" data-delay="2">

<div class="feature-icon"> </div>

<h3 data-i18n="feature_3_title">专业建议</h3>

<p data-i18n="feature_3_desc">提供最佳食用建议</p>

</div>

</div>

开始检测

```
</span>

        <span class="button-icon">    </span>

        <div class="button-glow"></div>

    </a>

</div>


<div class="scroll-indicator">

    <div class="mouse">

        <div class="wheel"></div>

    </div>

    <p>向下滚动了解更多</p>

</div>


<section class="info-section">

    <div class="info-card">

        <div class="info-icon">        </div>

        <h3 data-i18n="info_1_title">支持多种产品</h3>

        <p data-i18n="info_1_desc">黄瓜、西红柿、苹果等多种水果蔬菜的
成熟度检测</p>

    </div>

    <div class="info-card">

        <div class="info-icon">>📷 </div>

        <h3 data-i18n="info_2_title">快速检测</h3>

        <p data-i18n="info_2_desc">只需上传图片，几秒钟即可获得专业的
成熟度分析报告</p>

    </div>

    <div class="info-card">
```

```
<div class="info-icon"> </div>

<h3 data-i18n="info_3_title">隐私保护</h3>

<p data-i18n="info_3_desc">所有图片处理均在本地完成，保护您的
隐私安全</p>

</div>

</section>


<footer class="footer">

  <p data-i18n="footer_text">© 2024 成熟度检测系统 | 让每一口都是最
佳时刻</p>

</footer>

</div>


<script src="../js/i18n.js"></script>

<script src="../js/welcome.js"></script>

</body>

</html>
```

setting.py

```
<!DOCTYPE html>
```

```
<html lang="zh-CN">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>设置 - 成熟度检测系统</title>
```

```
    <link rel="stylesheet" href="../css/navbar.css">
```

```
    <link rel="stylesheet" href="../css/settings.css">
```

```
</head>
```

```
<body>
```

```
    <div class="particles" id="particles"></div>
```

```
    <nav class="navbar">
```

```
        <a href="/pages/welcome.html" class="nav-logo">          <span  
data-i18n="app_name">成熟度检测</span></a>
```

```
        <div class="nav-links">
```

```
            <a href="/pages/welcome.html" data-i18n="nav_home">首页</a>
```

```
            <a href="/pages/index.html" data-i18n="nav_detect">检测</a>
```

```
            <a href="/pages/result.html" data-i18n="nav_result">结果</a>
```

```
            <a href="/pages/models.html" data-i18n="nav_models">模型库</a>
```

```
            <a href="/pages/feedback.html" data-i18n="nav_feedback">反馈</a>
```

```
            <a href="/pages/history.html" data-i18n="nav_history">历史</a>
```

```
            <a href="/pages/profile.html" data-i18n="nav_profile">我的</a>
```

```
            <a href="/pages/settings.html" class="active" data-i18n="nav_settings">  
设置</a>
```

```
            <a href="/pages/ai-assistant.html">    AI 助手</a>
```

```
<a href="/pages/about.html" data-i18n="nav_about">关于</a>
    <a href="/pages/auth.html" data-i18n="nav_login">登录/注册</a>
</div>
</nav>
```

```
<div class="container">
    <div class="settings-header">
        <h1> 设置</h1>
        <p>自定义您的使用体验</p>
    </div>
```

```
<div class="settings-grid">
    <div class="settings-sidebar">
        <div class="sidebar-menu">
            <button class="menu-item active" data-section="account">
                <span class="menu-icon"> </span>
                <span class="menu-text">账号设置</span>
            </button>
            <button class="menu-item" data-section="notifications">
                <span class="menu-icon"> </span>
                <span class="menu-text">通知设置</span>
            </button>
            <button class="menu-item" data-section="privacy">
                <span class="menu-icon"> </span>
                <span class="menu-text">隐私设置</span>
            </button>
            <button class="menu-item" data-section="appearance">
```

```
        <span class="menu-icon">    </span>

        <span class="menu-text">外观设置</span>

    </button>

    <button class="menu-item" data-section="language">

        <span class="menu-icon">    </span>

        <span class="menu-text">语言设置</span>

    </button>

    <button class="menu-item" data-section="data">

        <span class="menu-icon">    </span>

        <span class="menu-text">数据管理</span>

    </button>

</div>

</div>

<div class="settings-content">

    <!-- 账号设置 -->

    <div class="settings-section active" id="account">

        <div class="section-header">

            <h2>    账号设置</h2>

            <p>管理您的账号信息</p>

        </div>

        <div class="setting-card">

            <div class="avatar-section">

                <div class="current-avatar">    </div>

                <div class="avatar-actions">

                    <button class="btn-primary">更换头像</button>

                </div>

            </div>

        </div>

    </div>

</div>
```

```

        <button class="btn-secondary">删除头像
    </button>

    </div>

</div>

</div>

<div class="setting-card">

    <h3>基本信息</h3>

    <div class="form-group">

        <label>昵称</label>

        <input type="text" value="水果达人" placeholder="请
        输入昵称">

    </div>

    <div class="form-group">

        <label>邮箱</label>

        <input type="email" value="user@example.com"
        placeholder="请输入邮箱">

    </div>

    <div class="form-group">

        <label>手机号</label>

        <input type="tel" value="138****8888" placeholder="
        请输入手机号">

    </div>

    <div class="form-group">

        <label>个人简介</label>

        <textarea rows="3" placeholder="介绍一下自
        己..."></textarea>

    </div>

```



```
        <button class="save-btn">保存更改</button>
    </div>
```

```

    <div class="setting-card danger-zone">
        <h3>    危险区域</h3>
        <div class="danger-item">
            <div>
                <span class="danger-title">修改密码</span>
                <span class="danger-desc">定期更改密码可以保
护账号安全</span>
            </div>
            <button class="btn-secondary">修改</button>
        </div>
        <div class="danger-item">
            <div>
                <span class="danger-title">注销账号</span>
                <span class="danger-desc">注销后所有数据将被
永久删除</span>
            </div>
            <button class="btn-danger">注销</button>
        </div>
    </div>
</div>
```

```
<!-- 通知设置 -->
<div class="settings-section" id="notifications">
    <div class="section-header">
```

```
<h2> 通知设置</h2>

<p>管理您的通知偏好</p>

</div>
```

```
<div class="setting-card">

  <h3>推送通知</h3>

  <div class="toggle-list">

    <div class="toggle-item">

      <div class="toggle-info">

        <span class="toggle-title">检测完成通知

        </span>

        <span class="toggle-desc">检测完成后发送

        通知</span>

      </div>

      <label class="toggle-switch">

        <input type="checkbox" checked>

        <span class="toggle-slider"></span>

      </label>

    </div>

    <div class="toggle-item">

      <div class="toggle-info">

        <span class="toggle-title">系统更新</span>

        <span class="toggle-desc">接收系统更新和

        新功能通知</span>

      </div>

      <label class="toggle-switch">

        <input type="checkbox" checked>

        <span class="toggle-slider"></span>

      </label>

    </div>

  </div>

</div>
```

```

        </label>
    </div>
    <div class="toggle-item">
        <div class="toggle-info">
            <span class="toggle-title">活动促销</span>
            <span class="toggle-desc">接收优惠活动和
促销信息</span>
        </div>
        <label class="toggle-switch">
            <input type="checkbox">
            <span class="toggle-slider"></span>
        </label>
    </div>
</div>
</div>
</div>

<div class="setting-card">
    <h3>邮件通知</h3>
    <div class="toggle-list">
        <div class="toggle-item">
            <div class="toggle-info">
                <span class="toggle-title">周报总结</span>
                <span class="toggle-desc">每周发送检测数
据总结</span>
            </div>
            <label class="toggle-switch">
                <input type="checkbox" checked>
            </label>
        </div>
    </div>
</div>

```

```

        <span class="toggle-slider"></span>
    </label>
</div>
<div class="toggle-item">
    <div class="toggle-info">
        <span class="toggle-title">账号安全提醒
    </span>
        <span class="toggle-desc">登录异常时发送
    邮件提醒</span>
    </div>
    <label class="toggle-switch">
        <input type="checkbox" checked>
        <span class="toggle-slider"></span>
    </label>
</div>
</div>
</div>
</div>

<!-- 隐私设置 -->
<div class="settings-section" id="privacy">
    <div class="section-header">
        <h2>    隐私设置</h2>
        <p>管理您的隐私和数据</p>
    </div>

    <div class="setting-card">
```

<h3>隐私选项</h3>

<div class="toggle-list">

<div class="toggle-item">

<div class="toggle-info">

公开检测记录

允许其他用户查

看您的检测记录

</div>

<label class="toggle-switch">

<input type="checkbox">

</label>

</div>

<div class="toggle-item">

<div class="toggle-info">

显示在排行榜

在排行榜中显示

您的用户名

</div>

<label class="toggle-switch">

<input type="checkbox" checked>

</label>

</div>

<div class="toggle-item">

<div class="toggle-info">

```

        <span class="toggle-title">允许数据分析
    </span>

    <span class="toggle-desc">允许我们使用您
    的数据改进服务</span>

    </div>

    <label class="toggle-switch">

        <input type="checkbox" checked>

        <span class="toggle-slider"></span>

    </label>

</div>

</div>

</div>

<div class="setting-card">

    <h3>数据下载</h3>

    <p class="card-desc">下载您的所有个人数据</p>

    <button class="btn-secondary">下载数据</button>

</div>

</div>

<!-- 外观设置 -->

<div class="settings-section" id="appearance">

    <div class="section-header">

        <h2>    外观设置</h2>

        <p>自定义界面外观</p>

    </div>

```

```
<div class="setting-card">

  <h3>主题模式</h3>

  <div class="theme-options">

    <label class="theme-option">

      <input type="radio" name="theme" value="dark"
checked>

      <span class="theme-preview dark">  </span>

      <span class="theme-label">深色模式</span>

    </label>

    <label class="theme-option">

      <input type="radio" name="theme"
value="light">

      <span class="theme-preview light">  </span>

      <span class="theme-label">浅色模式</span>

    </label>

    <label class="theme-option">

      <input type="radio" name="theme"
value="auto">

      <span class="theme-preview auto">  </span>

      <span class="theme-label">跟随系统</span>

    </label>

  </div>

</div>
```

```
<div class="setting-card">

  <h3>动画效果</h3>

  <div class="toggle-list">

    <div class="toggle-item">
```

```
<div class="toggle-info">

    <span class="toggle-title">页面动画</span>

    <span class="toggle-desc">启用页面过渡动

画效果</span>

</div>

<label class="toggle-switch">

    <input type="checkbox" checked>

    <span class="toggle-slider"></span>

</label>

</div>

<div class="toggle-item">

    <div class="toggle-info">

        <span class="toggle-title">粒子效果</span>

        <span class="toggle-desc">显示背景粒子动

画</span>

    </div>

    <label class="toggle-switch">

        <input type="checkbox" checked>

        <span class="toggle-slider"></span>

    </label>

</div>

</div>

</div>

</div>

<!-- 语言设置 -->

<div class="settings-section" id="language">
```



```
<div class="section-header">
```

```
    <h2>    语言设置</h2>
```

```
    <p>选择您偏好的语言</p>
```

```
</div>
```

```
<div class="setting-card">
```

```
    <h3>界面语言</h3>
```

```
    <div class="language-options">
```

```
        <label class="language-option">
```

```
            <input type="radio" name="language" value="zh"
checked>
```

```
            <span class="language-flag">    </span>
```

```
            <span class="language-name">简体中文</span>
```

```
        </label>
```

```
        <label class="language-option">
```

```
            <input type="radio" name="language"
value="en">
```

```
            <span class="language-flag">    </span>
```

```
            <span class="language-name">English</span>
```

```
        </label>
```

```
        <label class="language-option">
```

```
            <input type="radio" name="language"
value="ja">
```

```
            <span class="language-flag">    </span>
```

```
            <span class="language-name">日本語</span>
```

```
        </label>
```

```
    </div>
```

```
</div>
```

```
</div>
```

```
<!-- 数据管理 -->
```

```
<div class="settings-section" id="data">
```

```
  <div class="section-header">
```

```
    <h2>    数据管理</h2>
```

```
    <p>管理您的数据和存储</p>
```

```
  </div>
```

```
<div class="setting-card">
```

```
  <h3>存储使用情况</h3>
```

```
  <div class="storage-bar">
```

```
    <div class="storage-fill" style="width: 45%;"></div>
```

```
  </div>
```

```
  <div class="storage-info">
```

```
    <span>已使用 450MB / 1GB</span>
```

```
    <span>45%</span>
```

```
  </div>
```

```
</div>
```

```
<div class="setting-card">
```

```
  <h3>缓存管理</h3>
```

```
  <div class="cache-info">
```

```
    <span>当前缓存: 128MB</span>
```

```
    <button class="btn-secondary">清除缓存</button>
```

```
  </div>
```

```
</div>
```

```
<div class="setting-card">

  <h3>数据导出</h3>

  <p class="card-desc">导出您的检测历史记录</p>

  <div class="export-options">

    <button class="btn-secondary">导出为
JSON</button>

    <button class="btn-secondary">导出为
CSV</button>

    <button class="btn-secondary">导出为
PDF</button>

  </div>
</div>

<div class="setting-card danger-zone">

  <h3> 数据清除</h3>

  <div class="danger-item">

    <div>

      <span class="danger-title">清除所有检测记录
</span>

      <span class="danger-desc">此操作不可恢复
</span>

    </div>

    <button class="btn-danger">清除</button>

  </div>

</div>

</div>

</div>
```

</div>

</div>

<script src="../../js/i18n.js"></script>

<script src="../../js/settings.js"></script>

</body>

</html>

```
routespy

from flask import Blueprint, request, jsonify, Response

from flask_jwt_extended import create_access_token, jwt_required, get_jwt_identity

from werkzeug.security import check_password_hash

from models.models import db, User, Detection, Dataset

import os

import logging

import secrets

import requests

from datetime import datetime, timedelta

from ultralytics import YOLO

from utils.privacy import data_desensitizer, encryption_tool, access_control, audit_logger


# 配置日志
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)

logger = logging.getLogger(__name__)

api = Blueprint('api', __name__)


# 注册接口
@api.route('/register', methods=['POST'])
def register():
    try:
```

```
data = request.get_json()

# 脱敏处理后记录日志

if data:

    sanitized_data = data.copy()

    if 'email' in sanitized_data:

        sanitized_data['email'] =
data_desensitizer.desensitize_email(sanitized_data['email'])

    if 'password' in sanitized_data:

        sanitized_data['password'] = '***'

    logger.info(f"Register request: {sanitized_data}")

else:

    logger.error("No data provided")

    return jsonify({'message': '请提供注册信息'}), 400

required_fields = ['username', 'email', 'password', 'role']

for field in required_fields:

    if field not in data:

        logger.error(f"Missing field: {field}")

        return jsonify({'message': f'请提供{field}'}), 400

if User.query.filter_by(email=data['email']).first():

    logger.warning(f"Email already exists:
{data_desensitizer.desensitize_email(data['email'])}")

    return jsonify({'message': '邮箱已存在'}), 400

# 对密码进行加密处理

encrypted_password = encryption_tool.encrypt_data(data['password'])
```

```

new_user = User(
    username=data['username'],
    email=data['email'],
    password=encrypted_password, # 存储加密后的密码
    role=data['role']
)

db.session.add(new_user)

db.session.commit()


# 记录审计日志

audit_logger.log_data_access('system', 'user_registration', 'new user
registration')

logger.info(f"User registered successfully:
{data_desensitizer.desensitize_email(data['email'])}")

return jsonify({'message': '注册成功'}), 201

except Exception as e:

    logger.error(f"Error in register: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '注册失败，请稍后重试'}), 500


# 登录接口

@api.route('/login', methods=['POST'])

def login():

    try:

        data = request.get_json()

```

```
# 脱敏处理后记录日志

if data:

    sanitized_data = data.copy()

    if 'email' in sanitized_data:

        sanitized_data['email'] =
data_desensitizer.desensitize_email(sanitized_data['email'])

    if 'password' in sanitized_data:

        sanitized_data['password'] = '***'

    logger.info(f"Login request: {sanitized_data}")

else:

    logger.error("No data provided")

    return jsonify({'message': '请提供登录信息'}), 400


required_fields = ['email', 'password']

for field in required_fields:

    if field not in data:

        logger.error(f"Missing field: {field}")

        return jsonify({'message': f'请提供{field}'}), 400


user = User.query.filter_by(email=data['email']).first()

if not user:

    logger.warning(f"Invalid login attempt for email:
{data_desensitizer.desensitize_email(data['email'])}")

    return jsonify({'message': '邮箱或密码错误'}), 401


# 验证加密后的密码

decrypted_password = encryption_tool.decrypt_data(user.password)
```



```

        if decrypted_password != data['password']:

            logger.warning(f"Invalid login attempt for email:
{data_desensitizer.desensitize_email(data['email'])}")

            return jsonify({'message': '邮箱或密码错误'}), 401

        # 使用字符串类型的身份，避免 JWT 令牌验证问题

        access_token = create_access_token(identity=str(user.id))

        # 记录审计日志

        audit_logger.log_access(user.id, 'login', 'user account')

        logger.info(f"User logged in successfully:
{data_desensitizer.desensitize_email(data['email'])}")

        return jsonify({'access_token': access_token, 'user': {'id': user.id, 'username':
user.username, 'role': user.role}})

    except Exception as e:

        logger.error(f"Error in login: {str(e)}")

        import traceback

        traceback.print_exc()

        return jsonify({'message': '登录失败，请稍后重试'}), 500

# 检测接口（使用实际模型）

@api.route('/detect', methods=['POST'])

@api.route('/api/detect', methods=['POST'])

# @jwt_required()

def detect():

    try:

        # 尝试从 JWT 获取用户 ID，如果没有则使用默认消费者角色

        try:

            user_id = get_jwt_identity()

```

```
# 获取用户角色

user = User.query.filter_by(id=user_id).first()

user_role = user.role if user else 'consumer'

except:

    # 没有 JWT 令牌, 使用默认消费者角色

    user_id = 1  # 临时使用固定用户 ID

    user_role = 'consumer'


# 脱敏处理后记录日志

logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)}, User Role: {user_role}")


# 检查用户权限

if not access_control.check_permission(user_role, 'read'):

    logger.warning(f"Unauthorized access attempt by user: {data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限访问此功能'}), 403


# 获取上传的图片或图片 URL

logger.info(f"Files: {len(request.files) > 0}")

logger.info(f"Form keys: {list(request.form.keys())}")

logger.info(f"JSON data: {request.get_json()}")


# 记录所有请求参数

logger.info(f"All request form data: {dict(request.form)}")


image_path = None
```

```
# 支持微信小程序的 detectType 参数和 Web 端的 fruit_type 参数

detect_type = request.form.get('detectType') or request.form.get('fruit_type',
'unknown') or request.json.get('fruit_type', 'unknown')

# 将数字类型的 detectType 映射到对应的水果类型

type_mapping = {

    '0': 'cucumber', # 黄瓜

    '1': 'tomato',    # 番茄

    '2': 'apple'      # 苹果

}

fruit_type = type_mapping.get(detect_type, detect_type)

logger.info(f"Detect type: {detect_type}, Fruit type: {fruit_type}")


# 处理图片上传

if 'image' in request.files:

    image = request.files['image']

    logger.info(f"Image filename: {image.filename}")

    logger.info(f"Image content type: {image.content_type}")


# 保存图片

upload_folder = 'uploads'

if not os.path.exists(upload_folder):

    os.makedirs(upload_folder)

    logger.info(f"Created upload folder: {upload_folder}")


# 使用哈希值作为文件名，保护原始文件名信息

filename =

f"{datetime.now().strftime('%Y%m%d%H%M%S')}-{data_desensitizer.hash_identifier(image.fil
ename)[:8]}.jpg"
```

```

        image_path = os.path.join(upload_folder, filename)

        image.save(image_path)

        logger.info(f"Image saved to: {image_path}")

# 处理图片 URL
elif 'image_url' in request.form or (request.json and 'image_url' in request.json):

    image_url = request.form.get('image_url') or request.json.get('image_url')

    logger.info(f"Image URL: {image_url}")


# 下载图片

upload_folder = 'uploads'

if not os.path.exists(upload_folder):

    os.makedirs(upload_folder)

    logger.info(f"Created upload folder: {upload_folder}")


# 使用哈希值作为文件名

filename =

f"{datetime.now().strftime('%Y%m%d%H%M%S')}-{data_desensitizer.hash_identifier(image_u
rl)[:8]}.jpg"

image_path = os.path.join(upload_folder, filename)


# 下载图片

try:

    response = requests.get(image_url, timeout=10)

    response.raise_for_status()

    with open(image_path, 'wb') as f:

        f.write(response.content)

    logger.info(f"Image downloaded from URL and saved to:
{image_path}")

```

```

except Exception as e:

    logger.error(f"Failed to download image from URL: {str(e)}")

    return jsonify({'message': '下载图片失败, 请检查 URL 是否有效'}), 400

else:

    logger.error("No image or image_url provided")

    return jsonify({'message': '请上传图片或提供图片 URL'}), 400


# 加载模型并进行预测

current_dir = os.path.dirname(os.path.abspath(__file__))

model_path = os.path.join(current_dir,
'runs/detect/fruit_maturity_v2/weights/best.pt')

if os.path.exists(model_path):

    logger.info(f"Loading model: {model_path}")

    model = YOLO(model_path)

try:

    # 进行预测, 降低置信度阈值以检测更多对象

    results = model(image_path, verbose=False, conf=0.25) # 降低置
信度阈值到 0.25


    # 解析预测结果

    if results and len(results) > 0:

        result = results[0]

        boxes = result.boxes


    # 添加详细日志

    logger.info(f"Detection boxes count: {len(boxes)}")

```

```

if len(boxes) > 0:
    for i, box in enumerate(boxes):
        logger.info(f"Box {i}: class={int(box.cls)},
conf={float(box.conf):.3f}")

# 找到置信度最高的检测框

best_box = None

max_conf = 0

for box in boxes:
    conf = float(box.conf)
    if conf > max_conf:
        max_conf = conf
        best_box = box

# 如果没有找到检测框，返回错误

if best_box is None:
    logger.warning("No box found")
    return jsonify({'message': '检测失败：未检测到指定类型
的水果'}), 400

# 检查置信度是否足够高

if max_conf < 0.3: # 降低阈值到 0.3，提高检测灵敏度
    logger.warning(f"Low confidence: {max_conf}")
    return jsonify({'message': '检测失败：未检测到指定类型
的水果'}), 400

best_class = int(best_box.cls)

```

```
logger.info(f"Selected class ID: {best_class}, Confidence:  
{max_conf}")
```

```
logger.info(f"Detected class: {best_class}, model name:  
{model.names[best_class]}")
```

色分析

```
# 无论检测到什么类别, 都根据用户指定的水果类型进行颜色分析
```

```
# 这样即使模型检测错误, 也能根据颜色判断正确的成熟度
```

```
if fruit_type == 'cucumber':
```

```
    # 使用颜色分析来判断黄瓜成熟度
```

```
    import cv2
```

```
    import numpy as np
```

```
    img = cv2.imread(image_path)
```

```
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
    if best_box is not None:
```

```
        x1, y1, x2, y2 = map(int, best_box.xyxy[0].tolist())
```

```
        cucumber_region = img_rgb[y1:y2, x1:x2]
```

```
        if cucumber_region.size > 0:
```

```
            avg_color =
```

```
np.mean(cucumber_region.reshape(-1, 3), axis=0)
```

```
            avg_r, avg_g, avg_b = avg_color
```

```
            green_intensity = avg_g - avg_r
```

```
            max_color =
```

```
np.max(cucumber_region.reshape(-1, 3), axis=1).mean()
```

```
            min_color =
```

```

np.min(cucumber_region.reshape(-1, 3), axis=1).mean()

saturation = (max_color - min_color) /
(max_color + 1e-6) * 100

brightness = (avg_r + avg_g + avg_b) / 3

logger.info(f"Color analysis: R={avg_r:.1f},
G={avg_g:.1f}, B={avg_b:.1f}, green_intensity={green_intensity:.1f},
saturation={saturation:.1f}%, brightness={brightness:.1f}")

# 根据颜色判断黄瓜成熟度
if avg_g > avg_r + 30 and saturation > 25 and
brightness > 90:

    maturity_level = 3 # 成熟
elif avg_g > avg_r + 20 and brightness > 80:
    maturity_level = 2 # 半熟
elif brightness < 70:
    maturity_level = 1 # 未熟
else:
    maturity_level = 4 # 过熟

logger.info(f"Color-based maturity level:
{maturity_level}")

else:

    maturity_level = 1 # 默认未熟
else:
    maturity_level = 1 # 默认未熟
elif fruit_type == 'apple':

    # 使用颜色分析来判断苹果成熟度

```



```

import cv2

import numpy as np

img = cv2.imread(image_path)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

if best_box is not None:

    x1, y1, x2, y2 = map(int, best_box.xyxy[0].tolist())

    apple_region = img_rgb[y1:y2, x1:x2]

    if apple_region.size > 0:

        avg_color =
np.mean(apple_region.reshape(-1, 3), axis=0)

        avg_r, avg_g, avg_b = avg_color

        # 计算红色和绿色强度

        red_intensity = avg_r - avg_g

        # 计算颜色饱和度

        max_color = np.max(apple_region.reshape(-1,
3), axis=1).mean()

        min_color = np.min(apple_region.reshape(-1,
3), axis=1).mean()

        saturation = (max_color - min_color) /
(max_color + 1e-6) * 100

        # 计算亮度

        brightness = (avg_r + avg_g + avg_b) / 3

        logger.info(f"Apple color analysis:
R={avg_r:.1f}, G={avg_g:.1f}, B={avg_b:.1f}, red_intensity={red_intensity:.1f},

```

```
saturation={saturation:.1f}%, brightness={brightness:.1f}")
```

```
# 根据颜色判断苹果成熟度
```

```
if avg_r > avg_g + 30 and saturation > 25:
```

```
# 红色系苹果，成熟
```

```
maturity_level = 3
```

```
elif avg_r > avg_g + 10:
```

```
# 淡红色，半熟
```

```
maturity_level = 2
```

```
elif avg_g > avg_r + 10:
```

```
# 绿色系，未熟
```

```
maturity_level = 1
```

```
else:
```

```
# 其他情况，根据亮度判断
```

```
if brightness > 120:
```

```
maturity_level = 3 # 成熟
```

```
else:
```

```
maturity_level = 2 # 半熟
```

```
logger.info(f"Color-based apple maturity level:
```

```
{maturity_level}")
```

```
else:
```

```
maturity_level = 1 # 默认未熟
```

```
else:
```

```
maturity_level = 1 # 默认未熟
```

```
elif fruit_type == 'tomato':
```

```
# 使用颜色分析来判断番茄成熟度
```

```

import cv2

import numpy as np

img = cv2.imread(image_path)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

if best_box is not None:

    x1, y1, x2, y2 = map(int, best_box.xyxy[0].tolist())

    tomato_region = img_rgb[y1:y2, x1:x2]

    if tomato_region.size > 0:

        avg_color =
np.mean(tomato_region.reshape(-1, 3), axis=0)

        avg_r, avg_g, avg_b = avg_color

        # 计算红色强度

        red_intensity = avg_r - avg_g

        # 计算颜色饱和度

        max_color =
np.max(tomato_region.reshape(-1, 3), axis=1).mean()

        min_color =
np.min(tomato_region.reshape(-1, 3), axis=1).mean()

        saturation = (max_color - min_color) /
(max_color + 1e-6) * 100

        # 计算亮度

        brightness = (avg_r + avg_g + avg_b) / 3

        logger.info(f"Tomato color analysis:
R={avg_r:.1f}, G={avg_g:.1f}, B={avg_b:.1f}, red_intensity={red_intensity:.1f},

```

```

saturation={saturation:.1f}%, brightness={brightness:.1f}")

# 根据颜色判断番茄成熟度
if avg_r > avg_g + 40 and avg_r > avg_b + 30
and saturation > 30:

    # 深红色, 饱和度高 -> 成熟/过熟
    if brightness > 120:
        maturity_level = 3 # 成熟
    else:
        maturity_level = 4 # 过熟
elif avg_r > avg_g + 20 and avg_r > avg_b +
10:

    # 中等红色 -> 半熟
    maturity_level = 2
elif avg_g > avg_r or avg_b > avg_r:
    # 绿色或青色 -> 未熟
    maturity_level = 1
else:
    # 其他情况 -> 成熟
    maturity_level = 3

logger.info(f"Color-based tomato maturity
level: {maturity_level}")

else:

    maturity_level = 1 # 默认未熟

else:

    maturity_level = 1 # 默认未熟

else:

```

```
fruit_type = 'cucumber' # 默认

maturity_level = 2

logger.warning(f"Unexpected class ID: {best_class}")

confidence = round(max_conf, 2)

logger.info(f"Detection result: fruit_type={fruit_type},
class_id={best_class}, maturity={maturity_level}, confidence={confidence}")
```

计算糖度和保鲜期

根据成熟度等级映射糖度范围

```
def get_sugar_content(fruit_type, maturity_level):
```

```
    sugar_ranges = {
```

```
        'cucumber': {
```

```
            1: {'min': 1.0, 'max': 2.0},
```

```
            2: {'min': 2.0, 'max': 3.0},
```

```
            3: {'min': 3.0, 'max': 4.0},
```

```
            4: {'min': 4.0, 'max': 5.0}
```

```
        },
```

```
        'apple': {
```

```
            1: {'min': 8.0, 'max': 10.0},
```

```
            2: {'min': 10.0, 'max': 12.0},
```

```
            3: {'min': 12.0, 'max': 14.0},
```

```
            4: {'min': 14.0, 'max': 16.0}
```

```
        },
```

```
        'tomato': {
```

```
            1: {'min': 3.0, 'max': 4.0},
```

```
            2: {'min': 4.0, 'max': 5.0},
```

```
        3: {'min': 5.0, 'max': 6.0},
        4: {'min': 6.0, 'max': 7.0}
    }
}

return sugar_ranges.get(fruit_type,
sugar_ranges['apple']).get(maturity_level, {'min': 0, 'max': 0})
```

根据成熟度等级和水果类型计算保鲜期（天）

```
def get_shelf_life(fruit_type, maturity_level):
```

```
    shelf_life = {
```

```
        'cucumber': {
```

```
            1: 14,
```

```
            2: 10,
```

```
            3: 7,
```

```
            4: 3
```

```
        },
```

```
        'apple': {
```

```
            1: 60,
```

```
            2: 45,
```

```
            3: 30,
```

```
            4: 15
```

```
        },
```

```
        'tomato': {
```

```
            1: 14,
```

```
            2: 10,
```

```
            3: 7,
```

```
            4: 3
```

```

        }
    }

    return shelf_life.get(fruit_type,
shelf_life['apple']).get(maturity_level, 7)

    sugar_content = get_sugar_content(fruit_type,
maturity_level)

    shelf_life = get_shelf_life(fruit_type, maturity_level)

# 保存检测结果到数据库
new_detection = Detection(
    user_id=user_id,
    fruit_type=fruit_type,
    maturity_level=maturity_level,
    confidence=confidence,
    image_path=image_path
)

db.session.add(new_detection)

db.session.commit()

logger.info(f"Detection saved to database:
{new_detection.id}")

# 记录审计日志
audit_logger.log_data_access(user_id, 'detection', 'fruit
maturity detection')

# 构建响应数据，同时兼容 Web 端和微信小程序
response_data = {

```

```
        'fruit_type': fruit_type,
        'maturity_level': maturity_level,
        'confidence': confidence,
        'sugar_content': sugar_content,
        'shelf_life': shelf_life,
        'suggestion': get_suggestion(fruit_type, maturity_level,
user_role)
    }
```

```
# 为微信小程序添加特定格式
```

```
# 计算成熟度百分比
```

```
maturity_percentage = str(maturity_level * 25) + '%'
```

```
return jsonify({
    'success': True,
    'data': {
        'maturity': maturity_percentage,
        'suggestion': get_suggestion(fruit_type,
maturity_level, user_role),
        'sugarContent':
f"{sugar_content['min']}-{sugar_content['max']}%" if 'min' in sugar_content else None,
        'freshness': f"{shelf_life}天" if shelf_life else None
    },
    'result': response_data # 保持 Web 端兼容
})
else:
    # 未检测到目标，返回错误
    logger.warning("No objects detected")
```



```

        return jsonify({'success': False, 'message': '检测失败：未检测到指定类型的水果'}), 400

    else:

        # 预测失败，返回错误

        logger.warning("Prediction failed")

        return jsonify({'success': False, 'message': '检测失败：未检测到指定类型的水果'}), 400

    except Exception as e:

        # 模型预测失败，返回错误

        logger.error(f"Model prediction failed: {str(e)}")

        import traceback

        traceback.print_exc()

        return jsonify({'success': False, 'message': '检测失败：未检测到指定类型的水果'}), 400

    else:

        # 模型文件不存在，返回错误

        logger.error(f"Model file not found: {model_path}")

        return jsonify({'success': False, 'message': '检测失败：未检测到指定类型的水果'}), 400

    except Exception as e:

        logger.error(f"Error in detect: {str(e)}")

        import traceback

        traceback.print_exc()

        return jsonify({'success': False, 'message': '检测失败，请稍后重试'}), 500

# 获取历史记录

@api.route('/history', methods=['GET'])

def get_history():

```

```
try:

    # 尝试从 JWT 获取用户 ID, 如果没有则使用默认用户 ID

    try:

        user_id = get_jwt_identity()

        user_id_int = int(user_id)

    except:

        # 没有 JWT 令牌, 使用默认用户 ID

        user_id_int = 1

    # 脱敏处理后记录日志

    logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id_int)} accessing history")

    # 检查用户权限

    user = User.query.filter_by(id=user_id_int).first()

    user_role = user.role if user else 'consumer'

    if not access_control.check_permission(user_role, 'read'):

        logger.warning(f"Unauthorized access attempt by user: {data_desensitizer.hash_identifier(user_id_int)}")

        return jsonify({'message': '无权限访问此功能'}), 403

    # 使用整数类型的 user_id_int 进行查询

    detections =

Detection.query.filter_by(user_id=user_id_int).order_by(Detection.created_at.desc()).all()

    history = []

    for d in detections:
```

```

        history.append({
            'id': d.id,
            'fruit_type': d.fruit_type,
            'maturity_level': d.maturity_level,
            'confidence': d.confidence,
            'created_at': d.created_at.strftime('%Y-%m-%d %H:%M:%S')
        })

# 记录审计日志
audit_logger.log_data_access(user_id_int, 'history', 'access detection history')

return jsonify({'history': history})

except Exception as e:
    logger.error(f"Error in get_history: {str(e)}")
    import traceback
    traceback.print_exc()
    return jsonify({'message': '获取历史记录失败'}), 500

# 删除历史记录 API
@api.route('/history/<int:history_id>', methods=['DELETE'])
def delete_history(history_id):
    try:
        # 尝试从 JWT 获取用户 ID，如果没有则使用默认用户 ID
        try:
            user_id = get_jwt_identity()
            user_id_int = int(user_id)
        except:

```

```
# 没有 JWT 令牌, 使用默认用户 ID

user_id_int = 1


# 脱敏处理后记录日志

logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id_int)} deleting
history: {history_id}")


# 检查用户权限

user = User.query.filter_by(id=user_id_int).first()

user_role = user.role if user else 'consumer'


if not access_control.check_permission(user_role, 'delete'):

    logger.warning(f"Unauthorized delete attempt by user:
{data_desensitizer.hash_identifier(user_id_int)}")

    return jsonify({'message': '无权限执行此操作'}), 403


# 查找并删除历史记录

detection = Detection.query.filter_by(id=history_id, user_id=user_id_int).first()

if not detection:

    logger.warning(f"History not found: {history_id} for user:
{data_desensitizer.hash_identifier(user_id_int)}")

    return jsonify({'message': '历史记录不存在'}), 404


db.session.delete(detection)

db.session.commit()


# 记录审计日志

audit_logger.log_data_access(user_id_int, 'history', f'delete detection history:
```

```
{history_id})
```

```
    return jsonify({'message': '历史记录删除成功'}), 200
```

```
except Exception as e:
```

```
    logger.error(f"Error in delete_history: {str(e)}")
```

```
    import traceback
```

```
    traceback.print_exc()
```

```
    return jsonify({'message': '删除历史记录失败'}), 500
```

```
# 批量删除历史记录 API
```

```
@api.route('/history/batch-delete', methods=['POST'])
```

```
def batch_delete_history():
```

```
    try:
```

```
        # 尝试从 JWT 获取用户 ID，如果没有则使用默认用户 ID
```

```
    try:
```

```
        user_id = get_jwt_identity()
```

```
        user_id_int = int(user_id)
```

```
    except:
```

```
        # 没有 JWT 令牌，使用默认用户 ID
```

```
        user_id_int = 1
```

```
    data = request.get_json()
```

```
    if not data or 'ids' not in data:
```

```
        return jsonify({'message': '缺少必要参数'}), 400
```

```
    history_ids = data['ids']
```

```
    # 脱敏处理后记录日志
```

```
logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id_int)} batch  
deleting history: {history_ids}")
```

```
# 检查用户权限
```

```
user = User.query.filter_by(id=user_id_int).first()
```

```
user_role = user.role if user else 'consumer'
```

```
if not access_control.check_permission(user_role, 'delete'):
```

```
logger.warning(f"Unauthorized batch delete attempt by user:  
{data_desensitizer.hash_identifier(user_id_int)}")
```

```
return jsonify({'message': '无权限执行此操作'}), 403
```

```
# 批量删除历史记录
```

```
deleted_count =
```

```
Detection.query.filter_by(user_id=user_id_int).filter(Detection.id.in_(history_ids)).delete(sy  
nchronize_session=False)
```

```
db.session.commit()
```

```
# 记录审计日志
```

```
audit_logger.log_data_access(user_id_int, 'history', f'batch delete detection  
history: {deleted_count} records')
```

```
return jsonify({'message': f'成功删除 {deleted_count} 条历史记录'}), 200
```

```
except Exception as e:
```

```
logger.error(f"Error in batch_delete_history: {str(e)}")
```

```
import traceback
```

```
traceback.print_exc()
```

```
return jsonify({'message': '批量删除历史记录失败'}), 500
```

```
# 导出历史记录 API

@api.route('/history/export', methods=['GET'])

def export_history():

    try:

        # 尝试从 JWT 获取用户 ID，如果没有则使用默认用户 ID

        try:

            user_id = get_jwt_identity()

            user_id_int = int(user_id)

        except:

            # 没有 JWT 令牌，使用默认用户 ID

            user_id_int = 1

        # 脱敏处理后记录日志

        logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id_int)} exporting history")

        # 检查用户权限

        user = User.query.filter_by(id=user_id_int).first()

        user_role = user.role if user else 'consumer'

        if not access_control.check_permission(user_role, 'read'):

            logger.warning(f"Unauthorized export attempt by user: {data_desensitizer.hash_identifier(user_id_int)}")

            return jsonify({'message': '无权限执行此操作'}), 403

        # 获取历史记录
```

```
detections =  
Detection.query.filter_by(user_id=user_id_int).order_by(Detection.created_at.desc()).all()
```

```
# 构建 CSV 数据
```

```
import csv
```

```
import io
```

```
import datetime
```

```
output = io.StringIO()
```

```
writer = csv.writer(output)
```

```
writer.writerow(['ID', '水果类型', '成熟度等级', '置信度', '检测时间'])
```

```
fruit_names = {
```

```
    'cucumber': '黄瓜',
```

```
    'tomato': '番茄',
```

```
    'apple': '苹果'
```

```
}
```

```
for d in detections:
```

```
    writer.writerow([
```

```
        d.id,
```

```
        fruit_names.get(d.fruit_type, d.fruit_type),
```

```
        d.maturity_level,
```

```
        f"{d.confidence * 100:.1f}%",
```

```
        d.created_at.strftime('%Y-%m-%d %H:%M:%S')
```

```
    ])
```



```

        output.seek(0)

        # 记录审计日志
        audit_logger.log_data_access(user_id_int, 'history', 'export detection history')

        # 返回 CSV 文件
        return Response(output.getvalue(), mimetype='text/csv', headers={
            'Content-Disposition': f'attachment;
filename=history_export_{datetime.datetime.now().strftime("%Y%m%d_%H%M%S")}.csv'
        })

    except Exception as e:

        logger.error(f"Error in export_history: {str(e)}")

        import traceback

        traceback.print_exc()

        return jsonify({'message': '导出历史记录失败'}), 500

# 数据集管理接口
@api.route('/dataset/upload', methods=['POST'])
@jwt_required()
def upload_dataset():
    user_id = get_jwt_identity()

    # 脱敏处理后记录日志
    logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} uploading dataset")

    # 检查用户权限

```

```
user = User.query.filter_by(id=user_id).first()

user_role = user.role if user else 'consumer'

if not access_control.check_permission(user_role, 'write'):

    logger.warning(f"Unauthorized upload attempt by user:
{data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限上传数据集'}), 403

if 'image' not in request.files:

    return jsonify({'message': '请上传图片'}), 400

image = request.files['image']

fruit_type = request.form.get('fruit_type', 'unknown')

maturity_level = request.form.get('maturity_level', 1)

# 创建数据集目录

dataset_dir = 'dataset/raw_images'

if not os.path.exists(dataset_dir):

    os.makedirs(dataset_dir)

# 使用哈希值作为文件名，保护原始文件名信息

filename =

f"{datetime.now().strftime('%Y%m%d%H%M%S')}_{data_desensitizer.hash_identifier(image.filename[:8])}.jpg"

image_path = os.path.join(dataset_dir, filename)

image.save(image_path)

# 保存到数据库
```

```
new_dataset = Dataset(
    filename=filename,
    fruit_type=fruit_type,
    maturity_level=int(maturity_level),
    image_path=image_path,
    user_id=user_id
)
db.session.add(new_dataset)
db.session.commit()

# 记录审计日志
audit_logger.log_data_access(user_id, 'dataset', 'upload dataset')

return jsonify({
    'message': '数据集上传成功',
    'dataset': {
        'id': new_dataset.id,
        'filename': filename,
        'fruit_type': fruit_type,
        'maturity_level': maturity_level
    }
}), 201
```

```
@api.route('/dataset/list', methods=['GET'])
@jwt_required()
def get_dataset_list():
    user_id = get_jwt_identity()
```

```
# 脱敏处理后记录日志

logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} accessing dataset list")


# 检查用户权限

user = User.query.filter_by(id=user_id).first()

user_role = user.role if user else 'consumer'


if not access_control.check_permission(user_role, 'read'):

    logger.warning(f"Unauthorized access attempt by user: {data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限访问数据集'}), 403


# 获取查询参数

fruit_type = request.args.get('fruit_type')

maturity_level = request.args.get('maturity_level')

is_labeled = request.args.get('is_labeled')


# 构建查询

query = Dataset.query.filter_by(user_id=user_id)


if fruit_type:

    query = query.filter_by(fruit_type=fruit_type)


if maturity_level:

    query = query.filter_by(maturity_level=int(maturity_level))


if is_labeled is not None:
```

```

        query = query.filter_by(is_labeled=is_labeled.lower() == 'true')

    datasets = query.order_by(Dataset.created_at.desc()).all()

    dataset_list = []

    for d in datasets:

        dataset_list.append({

            'id': d.id,

            'filename': d.filename,

            'fruit_type': d.fruit_type,

            'maturity_level': d.maturity_level,

            'image_path': d.image_path,

            'is_labeled': d.is_labeled,

            'created_at': d.created_at.strftime('%Y-%m-%d %H:%M:%S')

        })

    # 记录审计日志

    audit_logger.log_data_access(user_id, 'dataset', 'access dataset list')

    return jsonify({'datasets': dataset_list, 'total': len(dataset_list)})

@api.route('/dataset/<int:dataset_id>', methods=['GET'])
@jwt_required()
def get_dataset(dataset_id):

    user_id = get_jwt_identity()

    # 脱敏处理后记录日志

```

```
logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} accessing dataset: {dataset_id}")
```

```
# 检查用户权限
```

```
user = User.query.filter_by(id=user_id).first()
```

```
user_role = user.role if user else 'consumer'
```

```
if not access_control.check_permission(user_role, 'read'):
```

```
    logger.warning(f"Unauthorized access attempt by user: {data_desensitizer.hash_identifier(user_id)}")
```

```
    return jsonify({'message': '无权限访问数据集'}), 403
```

```
dataset = Dataset.query.filter_by(id=dataset_id, user_id=user_id).first()
```

```
if not dataset:
```

```
    return jsonify({'message': '数据集不存在'}), 404
```

```
# 记录审计日志
```

```
audit_logger.log_data_access(user_id, 'dataset', f'access dataset {dataset_id}')
```

```
return jsonify({
```

```
    'id': dataset.id,
```

```
    'filename': dataset.filename,
```

```
    'fruit_type': dataset.fruit_type,
```

```
    'maturity_level': dataset.maturity_level,
```

```
    'image_path': dataset.image_path,
```

```
    'label_path': dataset.label_path,
```

```
        'is_labeled': dataset.is_labeled,  
        'created_at': dataset.created_at.strftime('%Y-%m-%d %H:%M:%S')  
    })
```

```
@api.route('/dataset/<int:dataset_id>', methods=['DELETE'])
```

```
@jwt_required()
```

```
def delete_dataset(dataset_id):
```

```
    user_id = get_jwt_identity()
```

```
    # 脱敏处理后记录日志
```

```
    logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} deleting dataset:  
{dataset_id}")
```

```
    # 检查用户权限
```

```
    user = User.query.filter_by(id=user_id).first()
```

```
    user_role = user.role if user else 'consumer'
```

```
    if not access_control.check_permission(user_role, 'delete'):
```

```
        logger.warning(f"Unauthorized delete attempt by user:  
{data_desensitizer.hash_identifier(user_id)}")
```

```
        return jsonify({'message': '无权限删除数据集'}), 403
```

```
    dataset = Dataset.query.filter_by(id=dataset_id, user_id=user_id).first()
```

```
    if not dataset:
```

```
        return jsonify({'message': '数据集不存在'}), 404
```

```
# 删除图片文件

if os.path.exists(dataset.image_path):

    os.remove(dataset.image_path)


# 删除标签文件

if dataset.label_path and os.path.exists(dataset.label_path):

    os.remove(dataset.label_path)


# 删除数据库记录

db.session.delete(dataset)

db.session.commit()


# 记录审计日志

audit_logger.log_data_access(user_id, 'dataset', f'delete dataset {dataset_id}')


return jsonify({'message': '数据集删除成功'})


@api.route('/dataset/<int:dataset_id>/label', methods=['PUT'])
@jwt_required()
def label_dataset(dataset_id):

    user_id = get_jwt_identity()


# 脱敏处理后记录日志

logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} labeling dataset: {dataset_id}")


# 检查用户权限
```



```
user = User.query.filter_by(id=user_id).first()

user_role = user.role if user else 'consumer'

if not access_control.check_permission(user_role, 'write'):

    logger.warning(f"Unauthorized label attempt by user:
{data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限更新数据集标签'}), 403

dataset = Dataset.query.filter_by(id=dataset_id, user_id=user_id).first()

if not dataset:

    return jsonify({'message': '数据集不存在'}), 404

data = request.get_json()

# 更新标签信息

if 'label_data' in data:

    # 保存标签文件

    labels_dir = 'dataset/raw_labels'

    if not os.path.exists(labels_dir):

        os.makedirs(labels_dir)

    label_filename = f"{os.path.splitext(dataset.filename)[0]}.txt"

    label_path = os.path.join(labels_dir, label_filename)

    with open(label_path, 'w') as f:

        f.write(data['label_data'])
```

```
dataset.label_path = label_path
```

```
dataset.is_labeled = True
```

```
if 'maturity_level' in data:
```

```
    dataset.maturity_level = int(data['maturity_level'])
```

```
db.session.commit()
```

```
# 记录审计日志
```

```
audit_logger.log_data_access(user_id, 'dataset', f'label dataset {dataset_id}')
```

```
return jsonify({'message': '标签更新成功'})
```

```
@api.route('/dataset/statistics', methods=['GET'])
```

```
@jwt_required()
```

```
def get_dataset_statistics():
```

```
    user_id = get_jwt_identity()
```

```
# 脱敏处理后记录日志
```

```
    logger.info(f"User ID: {data_desensitizer.hash_identifier(user_id)} accessing dataset statistics")
```

```
# 检查用户权限
```

```
user = User.query.filter_by(id=user_id).first()
```

```
user_role = user.role if user else 'consumer'
```

```
if not access_control.check_permission(user_role, 'read'):

    logger.warning(f"Unauthorized access attempt by user:
{data_desensitizer.hash_identifier(user_id)}")

    return jsonify({'message': '无权限访问数据集统计'}), 403


# 获取统计数据

total_datasets = Dataset.query.filter_by(user_id=user_id).count()

labeled_datasets = Dataset.query.filter_by(user_id=user_id, is_labeled=True).count()

unlabeled_datasets = total_datasets - labeled_datasets


# 按水果类型统计

fruit_stats = db.session.query(

    Dataset.fruit_type,

    db.func.count(Dataset.id)

).filter_by(user_id=user_id).group_by(Dataset.fruit_type).all()


fruit_counts = {fruit: count for fruit, count in fruit_stats}


# 按成熟度统计

maturity_stats = db.session.query(

    Dataset.maturity_level,

    db.func.count(Dataset.id)

).filter_by(user_id=user_id).group_by(Dataset.maturity_level).all()


maturity_counts = {str(level): count for level, count in maturity_stats}


# 记录审计日志
```

```
audit_logger.log_data_access(user_id, 'dataset', 'access dataset statistics')
```

```
return jsonify({  
    'total': total_datasets,  
    'labeled': labeled_datasets,  
    'unlabeled': unlabeled_datasets,  
    'by_fruit_type': fruit_counts,  
    'by_maturity_level': maturity_counts  
})
```

辅助函数：根据成熟度和用户角色返回建议

```
def get_suggestion(fruit_type, maturity_level, user_role):
```

```
    # 不同角色的建议
```

```
    role_suggestions = {
```

```
        'farmer': { # 农业工作者
```

```
            1: '适合采摘后长途运输',
```

```
            2: '适合本地销售',
```

```
            3: '适合立即销售',
```

```
            4: '不适合销售，建议用作其他用途'
```

```
        },
```

```
        'business': { # 商业人员
```

```
            1: '适合批量采购储存',
```

```
            2: '适合中等批量采购',
```

```
            3: '适合小批量采购',
```

```
            4: '不建议采购'
```

```
        },
```

```
        'consumer': { # 消费者
```

```
        1: '还需等待一段时间再食用',
        2: '可以食用， 但口感一般',
        3: '最佳食用时机',
        4: '不建议食用'
    }
}
```

```
# 获取对应角色的建议， 如果角色不存在则使用消费者的建议
suggestions = role_suggestions.get(user_role, role_suggestions['consumer'])
return suggestions.get(maturity_level, '未知')
```

```
# 忘记密码请求接口
```

```
@api.route('/forgot-password', methods=['POST'])
```

```
def forgot_password():
```

```
    try:
```

```
        data = request.get_json()
```

```
        if not data or 'email' not in data:
```

```
            logger.error("No email provided")
```

```
            return jsonify({'message': '请提供邮箱地址'}), 400
```

```
        email = data['email']
```

```
        # 脱敏处理后记录日志
```

```
        logger.info(f"Forgot password request for email: {data_desensitizer.desensitize_email(email)}")
```

```
        # 查找用户
```

```
user = User.query.filter_by(email=email).first()

if not user:

    logger.warning(f"Forgot password attempt for non-existent email:
{data_desensitizer.desensitize_email(email)}")

    return jsonify({'message': '邮箱不存在'}), 404


# 生成重置令牌

reset_token = secrets.token_urlsafe(32)

reset_token_expiry = datetime.utcnow() + timedelta(hours=1) # 1 小时过期


# 更新用户的重置令牌和过期时间

user.reset_token = reset_token

user.reset_token_expiry = reset_token_expiry

db.session.commit()


# 记录审计日志

audit_logger.log_access(user.id, 'forgot_password', 'password reset request')

logger.info(f"Reset token generated for user:
{data_desensitizer.desensitize_email(email)}")


# 这里应该发送邮件，包含重置链接

# 由于是本地开发，我们直接返回令牌

reset_link = f"http://127.0.0.1:5000/pages/auth.html?token={reset_token}"


return jsonify({

    'message': '重置链接已生成',

    'reset_link': reset_link,
```

```

        'reset_token': reset_token # 仅用于开发测试
    }, 200

except Exception as e:

    logger.error(f"Error in forgot_password: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500


# 验证重置令牌接口

@api.route('/reset-password/<token>', methods=['GET'])

def verify_reset_token(token):

    try:

        # 查找用户

        user = User.query.filter_by(reset_token=token).first()

        if not user:

            logger.warning(f"Invalid reset token: {token[:8]}...")

            return jsonify({'message': '无效的重置令牌'}), 404


        # 检查令牌是否过期

        if user.reset_token_expiry and user.reset_token_expiry < datetime.utcnow():

            logger.warning(f"Expired reset token for user: {user.id}")

            return jsonify({'message': '重置令牌已过期'}), 400


        logger.info(f"Reset token verified for user: {user.id}")

        return jsonify({

            'message': '令牌有效',

            'user_id': user.id,

```

```

        'email': data_desensitizer.desensitize_email(user.email)
    }), 200

except Exception as e:

    logger.error(f"Error in verify_reset_token: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500


# 重置密码接口

@api.route('/reset-password', methods=['POST'])

def reset_password():

    try:

        data = request.get_json()

        if not data or 'token' not in data or 'new_password' not in data:

            logger.error("Missing token or new password")

            return jsonify({'message': '缺少必要参数'}), 400

        token = data['token']

        new_password = data['new_password']

        # 查找用户

        user = User.query.filter_by(reset_token=token).first()

        if not user:

            logger.warning(f"Invalid reset token: {token}")

            return jsonify({'message': '无效的重置令牌'}), 404

```



```
# 检查令牌是否过期

if user.reset_token_expiry and user.reset_token_expiry < datetime.utcnow():

    logger.warning(f"Expired reset token: {token}")

    return jsonify({'message': '令牌已过期'}), 400


# 重置密码

encrypted_password = encryption_tool.encrypt_data(new_password)

user.password = encrypted_password

user.reset_token = None

user.reset_token_expiry = None

db.session.commit()


# 记录审计日志

audit_logger.log_access(user.id, 'reset_password', 'password reset')

logger.info(f"Password reset successful for user:
{data_desensitizer.desensitize_email(user.email)}")


return jsonify({'message': '密码重置成功'}), 200

except Exception as e:

    logger.error(f"Error in reset_password: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500


# 修改密码接口

@api.route('/change-password', methods=['POST'])

def change_password():
```

```
try:

    data = request.get_json()

    if not data or 'user_id' not in data or 'current_password' not in data or
'new_password' not in data:

        logger.error("Missing user_id, current_password, or new_password")

        return jsonify({'message': '缺少必要参数'}), 400

    user_id = data['user_id']

    current_password = data['current_password']

    new_password = data['new_password']

    # 查找用户

    user = User.query.get(user_id)

    if not user:

        logger.warning(f"User not found: {user_id}")

        return jsonify({'message': '用户不存在'}), 404

    # 验证当前密码

    decrypted_password = encryption_tool.decrypt_data(user.password)

    if decrypted_password != current_password:

        logger.warning(f"Incorrect current password for user: {user_id}")

        return jsonify({'message': '当前密码错误'}), 401

    # 更新密码

    encrypted_password = encryption_tool.encrypt_data(new_password)

    user.password = encrypted_password
```

```
db.session.commit()

# 记录审计日志

audit_logger.log_access(user_id, 'change_password', 'password changed')

logger.info(f"Password changed successful for user:
{data_desensitizer.desensitize_email(user.email)}")

return jsonify({'message': '密码修改成功'}), 200

except Exception as e:

    logger.error(f"Error in change_password: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500

# 直接修改密码接口（无需当前密码）

@api.route('/direct-change-password', methods=['POST'])

def direct_change_password():

    try:

        data = request.get_json()

        if not data or 'email' not in data or 'new_password' not in data:

            logger.error("Missing email or new_password")

            return jsonify({'message': '缺少必要参数'}), 400

        email = data['email']

        new_password = data['new_password']
```

```
# 查找用户

user = User.query.filter_by(email=email).first()

if not user:

    logger.warning(f"User not found for email:
{data_desensitizer.desensitize_email(email)}")

    return jsonify({'message': '邮箱不存在'}), 404


# 更新密码

encrypted_password = encryption_tool.encrypt_data(new_password)

user.password = encrypted_password

db.session.commit()


# 记录审计日志

audit_logger.log_access(user.id, 'direct_change_password', 'password changed
without current password')

logger.info(f"Direct password change successful for user:
{data_desensitizer.desensitize_email(user.email)}")


return jsonify({'message': '密码修改成功'}), 200

except Exception as e:

    logger.error(f"Error in direct_change_password: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500


# 获取 CSRF 令牌接口

@api.route('/csrf-token', methods=['GET'])

def get_csrf_token():
```

```
from flask_wtf.csrf import generate_csrf

return jsonify({'csrf_token': generate_csrf()})


# 获取用户信息接口

@api.route('/user-info', methods=['GET'])
@jwt_required()
def get_user_info():
    try:
        user_id = get_jwt_identity()

        # 查找用户
        user = User.query.get(user_id)

        if not user:
            logger.warning(f"User not found: {user_id}")
            return jsonify({'message': '用户不存在'}), 404

        # 构建用户信息
        user_info = {
            'id': user.id,
            'username': user.username,
            'email': user.email,
            'role': user.role,
            'roleName': {
                'farmer': '农业工作者',
                'consumer': '消费者',
                'business': '商业人员'
            }.get(user.role, user.role)
        }
```

```
}
```

```
        logger.info(f"User info retrieved for user:  
{data_desensitizer.desensitize_email(user.email)}")
```

```
        return jsonify({'user': user_info}), 200
```

```
    except Exception as e:
```

```
        logger.error(f"Error in get_user_info: {str(e)}")
```

```
        import traceback
```

```
        traceback.print_exc()
```

```
        return jsonify({'message': '处理失败，请稍后重试'}), 500
```

```
# 更新用户信息接口
```

```
@api.route('/update-user', methods=['PUT'])
```

```
@jwt_required()
```

```
def update_user():
```

```
    try:
```

```
        user_id = get_jwt_identity()
```

```
        data = request.get_json()
```

```
    # 查找用户
```

```
    user = User.query.get(user_id)
```

```
    if not user:
```

```
        logger.warning(f"User not found: {user_id}")
```

```
        return jsonify({'message': '用户不存在'}), 404
```

```
    # 更新用户信息
```

```
    if 'username' in data:
```

```
        user.username = data['username']

    db.session.commit()

    # 记录审计日志
    audit_logger.log_access(user_id, 'update_user', 'user information updated')

    logger.info(f"User info updated for user:
{data_desensitizer.desensitize_email(user.email)}")

    return jsonify({'message': '用户信息更新成功'}), 200
except Exception as e:

    logger.error(f"Error in update_user: {str(e)}")

    import traceback

    traceback.print_exc()

    return jsonify({'message': '处理失败，请稍后重试'}), 500
```

```
ai_routes.py

from flask import Blueprint, request, jsonify

import logging

import json

from datetime import datetime


ai_api = Blueprint('ai_api', __name__)


logger = logging.getLogger(__name__)


# 水果知识库

FRUIT_KNOWLEDGE_BASE = {

    'cucumber': {

        'name': '黄瓜',

        'maturity_levels': {

            '1': {

                'name': '生',

                'description': '黄瓜完全未成熟，颜色深绿，质地坚硬',

                'storage': '常温保存，避免阳光直射，可保存 3-5 天',

                'harvest': '不建议采摘，需要继续生长',

                'market': '不适合市场销售',

                'processing': '可制作腌黄瓜'

            },

            '2': {

                'name': '半成熟',

                'description': '黄瓜开始成熟，颜色浅绿，质地较硬',

                'storage': '冷藏保存，温度 10-12°C，可保存 7-10 天',
```



```
      'harvest': '可以采摘，但建议再等待 1-2 天',
      'market': '适合批发市场销售',
      'processing': '适合制作沙拉和凉拌菜'
    },
    '3': {
      'name': '成熟',
      'description': '黄瓜完全成熟，颜色翠绿，质地适中',
      'storage': '冷藏保存，温度 8-10°C，可保存 10-14 天',
      'harvest': '最佳采摘时机',
      'market': '适合零售市场，价格最优',
      'processing': '适合各种烹饪方式'
    },
    '4': {
      'name': '过熟',
      'description': '黄瓜过度成熟，颜色发黄，质地变软',
      'storage': '立即食用，不宜储存',
      'harvest': '应立即采摘',
      'market': '价格较低，需快速销售',
      'processing': '适合制作黄瓜汁或汤'
    }
  },
  'planting_tips': [
    '黄瓜喜欢温暖湿润的环境，适宜温度 20-30°C',
    '需要充足的阳光，每天至少 6 小时直射光',
    '土壤要求排水良好，pH 值 6.0-7.0',
    '定期浇水，保持土壤湿润但不过湿',
    '生长期需要充足的营养，建议使用有机肥料'
```

```
    ],  
  
    'common_questions': {  
  
        '如何判断成熟度': '观察颜色：成熟的黄瓜颜色翠绿，有光泽；触摸质地：成熟的黄瓜质地适中，不过硬也不过软；检查刺：成熟的黄瓜刺比较完整，不脱落。',  
  
        '最佳储存温度': '黄瓜的最佳储存温度是 8-12°C，相对湿度 85-90%。避免与乙烯产生的水果（如苹果、香蕉）一起储存。',  
  
        '如何延长保鲜期': '1. 用保鲜膜包裹，减少水分流失；2. 避免清洗，食用前再清洗；3. 保持适宜的温度和湿度；4. 定期检查，及时处理受损的黄瓜。',  
  
        '好处': '黄瓜富含水分、维生素 C、钾等营养成分，具有以下好处：1. 补水保湿，美容养颜；2. 促进消化，改善便秘；3. 降低血压，保护心脏；4. 增强免疫力；5. 减肥瘦身，热量低。'  
  
    }  
  
},  
  
'tomato': {  
  
    'name': '番茄',  
  
    'maturity_levels': {  
  
        '1': {  
  
            'name': '生',  
  
            'description': '番茄完全未成熟，颜色深绿，质地坚硬',  
  
            'storage': '常温保存，可催熟，保存时间较长',  
  
            'harvest': '不建议采摘，需要继续生长',  
  
            'market': '不适合市场销售',  
  
            'processing': '可制作青番茄炒菜'  
  
        },  
  
        '2': {  
  
            'name': '半成熟',  
  
            'description': '番茄开始成熟，颜色开始变红，质地较硬',  
  
            'storage': '常温催熟，或冷藏保存',  
  
        },  
  
    },  
  
}
```

```
        'harvest': '可以采摘，适合运输',
        'market': '适合长途运输销售',
        'processing': '适合制作番茄酱和罐头'
    },
    '3': {
        'name': '成熟',
        'description': '番茄完全成熟，颜色鲜红，质地适中',
        'storage': '常温保存，避免阳光直射，可保存 5-7 天',
        'harvest': '最佳采摘时机',
        'market': '适合零售市场，口感最佳',
        'processing': '适合生食和烹饪'
    },
    '4': {
        'name': '过熟',
        'description': '番茄过度成熟，颜色深红，质地变软',
        'storage': '立即食用或加工，不宜储存',
        'harvest': '应立即采摘',
        'market': '价格较低，需快速销售',
        'processing': '适合制作番茄汁、汤或果酱'
    }
},
'planting_tips': [
    '番茄喜欢温暖的环境，适宜温度 18-28°C',
    '需要充足的阳光，每天至少 8 小时直射光',
    '土壤要求肥沃排水良好，pH 值 6.0-7.0',
    '需要支架支撑，防止果实触地腐烂',
    '定期修剪侧枝，促进主枝生长'
```

```
    ],  
    'common_questions': {  
        '如何判断成熟度': '观察颜色：成熟的番茄颜色鲜红均匀；触摸质地：成熟的番茄质地适中，有弹性；检查果蒂：成熟的番茄果蒂容易脱落。',  
        '最佳储存温度': '番茄的最佳储存温度是 12-15°C，相对湿度 85-90%。完全成熟的番茄可以常温保存，未成熟的可以催熟。',  
        '如何延长保鲜期': '1. 避免与乙烯敏感的蔬菜一起储存；2. 储存时果蒂朝下，减少水分流失；3. 避免阳光直射；4. 不要在完全成熟前放入冰箱。',  
        '好处': '番茄富含番茄红素、维生素 C、维生素 E 等营养成分，具有以下好处：1. 抗氧化，延缓衰老；2. 保护心血管健康；3. 增强免疫力；4. 促进消化；5. 预防癌症。'  
    }  
},  
    'apple': {  
        'name': '苹果',  
        'maturity_levels': {  
            '1': {  
                'name': '生',  
                'description': '苹果完全未成熟，颜色青绿，质地坚硬',  
                'storage': '冷藏保存，可储存较长时间',  
                'harvest': '不建议采摘，需要继续生长',  
                'market': '不适合市场销售',  
                'processing': '可制作苹果醋'  
            },  
            '2': {  
                'name': '半成熟',  
                'description': '苹果开始成熟，颜色开始变化，质地较硬',  
                'storage': '冷藏保存，可储存 2-3 个月',
```

```
        'harvest': '可以采摘，适合储存',
        'market': '适合批发市场销售',
        'processing': '适合制作苹果干和果酱'
    },
    '3': {
        'name': '成熟',
        'description': '苹果完全成熟，颜色鲜艳，质地适中',
        'storage': '冷藏保存，可储存 1-2 个月',
        'harvest': '最佳采摘时机',
        'market': '适合零售市场，口感最佳',
        'processing': '适合生食和烹饪'
    },
    '4': {
        'name': '过熟',
        'description': '苹果过度成熟，颜色深，质地变软',
        'storage': '立即食用或加工，不宜储存',
        'harvest': '应立即采摘',
        'market': '价格较低，需快速销售',
        'processing': '适合制作苹果派和果汁'
    }
},
'planting_tips': [
    '苹果喜欢冷凉的环境，适宜温度 15-25°C',
    '需要充足的阳光，每天至少 6 小时直射光',
    '土壤要求深厚肥沃，排水良好，pH 值 6.0-7.5',
    '需要定期修剪，促进通风透光',
    '注意病虫害防治，定期喷洒农药'
```

```

    ],

    'common_questions': {

        '如何判断成熟度': '观察颜色：成熟的苹果颜色鲜艳有光泽；触摸质地：成熟的苹果质地适中，不过硬也过软；闻气味：成熟的苹果有浓郁的果香。',

        '最佳储存温度': '苹果的最佳储存温度是 0-4°C，相对湿度 90-95%。可以储存数月，但要注意避免与有强烈气味的食物一起储存。',

        '如何延长保鲜期': '1. 单独包装，避免相互挤压；2. 保持适宜的温度和湿度；3. 定期检查，及时处理受损的苹果；4. 避免与乙烯产生的水果一起储存。',

        '好处': '苹果富含膳食纤维、维生素 C、钾等营养成分，具有以下好处：1. 促进肠道健康，预防便秘；2. 降低胆固醇，保护心脏；3. 增强免疫力；4. 抗氧化，延缓衰老；5. 有助于控制体重。'

    }

}

}

```

```

@api.route('/ai/chat', methods=['POST'])

def ai_chat():

    try:

        data = request.get_json()

        message = data.get('message', "").strip()

        if not message:

            return jsonify({'message': '请输入问题'}), 400

        logger.info(f"AI Chat Request: {message}")

        response = generate_ai_response(message)

```

```

        return jsonify({'response': response})

except Exception as e:

    logger.error(f"AI Chat Error: {e}")

    return jsonify({'message': '处理请求时出错'}), 500


@api.route('/ai/image-qa', methods=['POST'])
def ai_image_qa():

    try:

        data = request.get_json()

        image = data.get('image', "")

        question = data.get('question', "").strip()

        if not image:

            return jsonify({'message': '请上传图片'}), 400

        if not question:

            return jsonify({'message': '请输入问题'}), 400

        logger.info(f"AI Image QA Request: {question}")

        response = generate_image_qa_response(question)

        return jsonify({'response': response})

    except Exception as e:

        logger.error(f"AI Image QA Error: {e}")

        return jsonify({'message': '处理请求时出错'}), 500

```

```
@ai_api.route('/ai/recommendation', methods=['POST'])

def ai_recommendation():

    try:

        data = request.get_json()

        fruit_type = data.get('fruit_type', '')

        maturity_level = data.get('maturity_level', '')

        recommendation_type = data.get('recommendation_type', 'all')

        if not fruit_type or not maturity_level:

            return jsonify({'message': '请提供水果类型和成熟度'}), 400

        logger.info(f"AI Recommendation Request: {fruit_type}, {maturity_level}, {recommendation_type}")

        recommendations = generate_recommendations(fruit_type, maturity_level, recommendation_type)

        return jsonify({'recommendations': recommendations})

    except Exception as e:

        logger.error(f"AI Recommendation Error: {e}")

        return jsonify({'message': '处理请求时出错'}), 500

def generate_ai_response(message):

    message_lower = message.lower()
```



```
# 首先提取水果类型
```

```
detected_fruit = None
```

```
for fruit_key, fruit_info in FRUIT_KNOWLEDGE_BASE.items():
```

```
    fruit_name = fruit_info['name']
```

```
    if fruit_name in message or fruit_key in message:
```

```
        detected_fruit = fruit_info
```

```
        break
```

```
# 1. 采摘相关问题
```

```
    if '采摘' in message or '收获' in message or ('什么时候' in message and '食用' not in
message and '口感' not in message and '吃' not in message):
```

```
        if '运输' in message or '运' in message:
```

```
            if detected_fruit:
```

```
                # 针对特定水果的运输采摘建议
```

```
                return f'''
```

```
                <p><strong>{detected_fruit['name']}运输采摘建议： </strong></p>
```

```
                <p>如果需要运输, 建议在<strong>2 级 - 半成熟</strong>时采摘,
此时水果： </p>
```

```
                <ul>
```

```
                    <li>已经开始成熟, 品质有保证</li>
```

```
                    <li>质地较硬, 不易在运输过程中受损</li>
```

```
                    <li>有较长的货架期</li>
```

```
                </ul>
```

```
                <p>具体来说,
```

```
{detected_fruit['maturity_levels']['2']['description']}</p>
```

```
                '''
```

```
            else:
```

```
                # 通用运输采摘建议
```

```

        return ""

        <p><strong>运输采摘建议： </strong></p>

        <p>对于需要运输的水果， 建议在<strong>2 级 - 半成熟</strong>
        时采摘， 原因： </p>

        <ul>

            <li><strong>硬度适中</strong>： 半成熟的水果质地较硬， 不
            易在运输过程中受损</li>

            <li><strong>货架期长</strong>： 半成熟的水果可以在运输和
            储存过程中继续成熟</li>

            <li><strong>品质保证</strong>： 已经开始成熟， 到达目的地
            后正好达到最佳食用状态</li>

        </ul>

        <p>不同水果的具体采摘时间可能有所差异， 您可以指定具体水果类
        型获取更详细的建议。</p>

        ""

    else:

        if detected_fruit:

            # 针对特定水果的最佳采摘时机

            return f""

            <p><strong>{detected_fruit['name']}最佳采摘时机： </strong></p>

            <p><strong>最佳采摘期</strong>：
            {detected_fruit['maturity_levels']['3']['description']}</p>

            <p><strong>采摘判断标准</strong>：
            {detected_fruit['common_questions']['如何判断成熟度']}</p>

            <p><strong>不同成熟度采摘建议</strong>： </p>

            <ul>

                <li><strong>1 级 - 生</strong>：
                {detected_fruit['maturity_levels']['1']['harvest']}</li>

                <li><strong>2 级 - 半成熟</strong>：

```

```

{detected_fruit['maturity_levels']['2']['harvest']}</li>

        <li><strong>3 级 - 成熟</strong>:
{detected_fruit['maturity_levels']['3']['harvest']}</li>

        <li><strong>4 级 - 过熟</strong>:
{detected_fruit['maturity_levels']['4']['harvest']}</li>

    </ul>

    '''

else:

    # 通用采摘时机建议

    return ''

    <p><strong>水果最佳采摘时机: </strong></p>

    <p>不同水果的最佳采摘时机有所不同，但一般遵循以下原则: </p>

    <ul>

        <li><strong>外观特征</strong>: 观察水果的颜色、大小、光
泽度</li>

        <li><strong>质地触摸</strong>: 成熟的水果质地适中，有弹
性</li>

        <li><strong>气味闻辨</strong>: 成熟的水果有浓郁的果香
</li>

        <li><strong>果蒂状态</strong>: 成熟的水果果蒂容易脱落
</li>

    </ul>

    <p>如果是为了运输，建议在半成熟期采摘；如果是即时食用，建议
在完全成熟期采摘。</p>

    <p>您可以指定具体水果类型获取更详细的采摘建议。</p>

    '''

# 2. 食用口感相关问题

if '食用' in message or '口感' in message or '味道' in message or '吃' in message:

```

```
if '三个品种' in message or '三种' in message:
```

```
# 回答所有三种水果的最佳食用时机
```

```
return ""
```

```
<p><strong>三种水果的最佳食用时机：</strong></p>
```

```
<p><strong>黄瓜：</strong></p>
```

```
<ul>
```

```
<li><strong>最佳食用期</strong>：3 级 - 成熟时，此时黄瓜颜色  
翠绿，质地适中，口感脆嫩多汁</li>
```

```
<li><strong>口感特点</strong>：脆爽可口，水分充足，适合生食、  
凉拌或炒菜</li>
```

```
<li><strong>食用建议</strong>：成熟的黄瓜可以直接食用，也可  
以制作各种菜肴</li>
```

```
</ul>
```

```
<p><strong>番茄：</strong></p>
```

```
<ul>
```

```
<li><strong>最佳食用期</strong>：3 级 - 成熟时，此时番茄颜色  
鲜红，质地柔软，口感甜美多汁</li>
```

```
<li><strong>口感特点</strong>：酸甜适中，果肉饱满，适合生食、  
炒菜或制作番茄酱</li>
```

```
<li><strong>食用建议</strong>：成熟的番茄可以直接食用，也可  
以用于各种烹饪</li>
```

```
</ul>
```

```
<p><strong>苹果：</strong></p>
```

```
<ul>
```

```
<li><strong>最佳食用期</strong>：3 级 - 成熟时，此时苹果颜色  
鲜艳，质地脆爽，口感甜美</li>
```

口感特点: 清脆多汁, 甜度适中, 适合生食或制作甜点

食用建议: 成熟的苹果可以直接食用, 也可以制作苹果派、苹果汁等

'''

elif detected_fruit:

针对特定水果的最佳食用时机

fruit_name = detected_fruit['name']

return f'''

<p>{fruit_name}最佳食用时机: </p>

<p>最佳食用期: 3 级 - 成熟时, 此时 {detected_fruit['maturity_levels']['3']['description']}</p>

<p>口感特点: </p>

完全成熟的{fruit_name}口感最佳

甜度适中, 风味浓郁

质地柔软 (番茄) 或脆爽 (黄瓜、苹果)

<p>食用建议: </p>

成熟的{fruit_name}可以直接食用

也可以用于烹饪或制作各种菜肴

建议在采摘后尽快食用, 以保持最佳口感

'''

else:

通用食用口感建议

```

return ""

<p><strong>水果最佳食用时机：</strong></p>

<p>一般来说，水果在<strong>3 级 - 成熟</strong>时食用口感最佳，
此时：</p>

<ul>

    <li>糖分含量达到最高</li>

    <li>风味最浓郁</li>

    <li>质地适中，口感最好</li>

</ul>

<p>不同水果的具体最佳食用时机：</p>

<ul>

    <li><strong>黄瓜</strong>：成熟时颜色翠绿，口感脆嫩多汁</li>

    <li><strong>番茄</strong>：成熟时颜色鲜红，口感甜美多汁</li>

    <li><strong>苹果</strong>：成熟时颜色鲜艳，口感脆爽甜美</li>

</ul>

<p>建议在水果完全成熟后尽快食用，以获得最佳口感体验。</p>
'''

```

2. 检测相关的问题

if '检测' in message or '成熟度' in message:

if '如何判断' in message or '怎么判断' in message:

if detected_fruit:

return f'''

<p>如何判断{detected_fruit['name']}成熟度：

</p>

<p>{detected_fruit['common_questions']['如何判断成熟度']}</p>

<p>您也可以使用我们的 AI 检测系统，上传{detected_fruit['name']}
 图片即可自动判断成熟度！</p>

'''

else:

return '''

<p>判断水果成熟度的方法有几种: </p>

颜色观察: 成熟的水果通常有特定的颜色特征

质地触摸: 成熟的水果质地适中, 不过硬也不过软

气味闻辨: 成熟的水果有浓郁的果香

大小检查: 成熟的水果通常达到标准大小

<p>您可以使用我们的 AI 检测系统, 上传水果图片即可自动判断成熟度! </p>

'''

elif '等级' in message or '级别' in message:

return '''

<p>成熟度等级分为 4 级: </p>

1 级 - 生: 完全未成熟, 需要继续生长

2 级 - 半成熟: 开始成熟, 适合运输

3 级 - 成熟: 完全成熟, 最佳食用时机

4 级 - 过熟: 过度成熟, 需要立即食用或加工

<p>不同成熟度的水果有不同的储存和销售建议, 您可以在"智能推荐"功能中获取详细建议。</p>

```
'''
```

3. 特定水果的问题

```
if detected_fruit:
```

```
    fruit_name = detected_fruit['name']
```

```
    if '储存' in message or '保存' in message:
```

```
        return f'''
```

```
        <p><strong>{fruit_name}储存建议: </strong></p>
```

```
        <p>{detected_fruit['common_questions']['最佳储存温度']}</p>
```

```
        <p>{detected_fruit['common_questions']['如何延长保鲜期']}</p>
```

```
        '''
```

```
    elif '种植' in message or '生长' in message:
```

```
        tips = '<br>'.join([f'<li>{tip}</li>' for tip in detected_fruit['planting_tips']])
```

```
        return f'''
```

```
        <p><strong>{fruit_name}种植技巧: </strong></p>
```

```
        <ul>{tips}</ul>
```

```
        '''
```

```
    elif '好处' in message or '营养' in message or '价值' in message:
```

```
        if '好处' in detected_fruit['common_questions']:
```

```
            return f'''
```

```
            <p><strong>{fruit_name}的好处: </strong></p>
```

```
            <p>{detected_fruit['common_questions']['好处']}</p>
```

```
            '''
```

```
    elif '市场' in message or '销售' in message:
```

```
        return f'''
```

```
        <p><strong>{fruit_name}市场销售建议: </strong></p>
```

```
        <ul>
```



```

        <li><strong>1 级 - 生</strong>:
{detected_fruit['maturity_levels']['1']['market']}</li>

        <li><strong>2 级 - 半成熟</strong>:
{detected_fruit['maturity_levels']['2']['market']}</li>

        <li><strong>3 级 - 成熟</strong>:
{detected_fruit['maturity_levels']['3']['market']}</li>

        <li><strong>4 级 - 过熟</strong>:
{detected_fruit['maturity_levels']['4']['market']}</li>

    </ul>

```

```

'''

```

```

elif '加工' in message or '烹饪' in message:

```

```

    return f'''

```

```

        <p><strong>{fruit_name}加工建议: </strong></p>

```

```

    <ul>

```

```

        <li><strong>1 级 - 生</strong>:
{detected_fruit['maturity_levels']['1']['processing']}</li>

```

```

        <li><strong>2 级 - 半成熟</strong>:
{detected_fruit['maturity_levels']['2']['processing']}</li>

```

```

        <li><strong>3 级 - 成熟</strong>:
{detected_fruit['maturity_levels']['3']['processing']}</li>

```

```

        <li><strong>4 级 - 过熟</strong>:
{detected_fruit['maturity_levels']['4']['processing']}</li>

```

```

    </ul>

```

```

'''

```

4. 系统相关问题

```

if '系统' in message or '功能' in message or '如何使用' in message:

```

```

    return '''

```

```

        <p>我们的系统提供以下功能: </p>

```

```
<ul>

    <li><strong>成熟度检测</strong>: 上传水果图片, 自动分析成熟度</li>

    <li><strong>智能推荐</strong>: 根据检测结果提供个性化建议</li>

    <li><strong>AI 助手</strong>: 解答关于水果的各种问题</li>

    <li><strong>图片问答</strong>: 针对水果图片进行智能问答</li>

</ul>

<p>使用方法: </p>

<ol>

    <li>在首页选择"开始检测"</li>

    <li>上传水果图片</li>

    <li>选择水果类型</li>

    <li>点击"开始检测"按钮</li>

    <li>查看检测结果和智能建议</li>

</ol>

'''
```

5. 问候和通用问题

```
if '你好' in message or '您好' in message or 'hi' in message_lower or 'hello' in
message_lower:
```

```
    return ""

    <p>您好! 我是水果成熟度检测系统的 AI 助手, 很高兴为您服务。</p>

    <p>我可以帮您: </p>

    <ul>

        <li>解答关于水果成熟度的问题</li>

        <li>提供水果种植和储存建议</li>

        <li>回答关于水果营养价值的问题</li>

        <li>指导您如何使用我们的检测系统</li>
```


<p>请问有什么可以帮助您的？ </p>

'''

6. 通用回答

general_responses = [

'''

<p>您好！我是 AI 助手，可以帮您解答关于水果成熟度检测、种植、储存等各种问题。</p>

<p>您可以问我： </p>

如何判断水果的成熟度？

不同成熟度水果的储存方法？

什么时候采摘水果最好？

水果的营养价值有哪些？

'''

'''

<p>关于成熟度检测，我们的系统使用先进的 AI 技术，可以准确识别黄瓜、番茄、苹果的成熟度。</p>

<p>检测过程： </p>

上传水果图片

选择水果类型

AI 自动分析

获得成熟度结果和建议


```

'''
'''

<p>对于您的问题， 我建议您： </p>

<ul>

    <li>使用我们的"智能推荐"功能， 根据检测结果获取个性化建议</li>

    <li>上传水果图片， 使用"图片问答"功能获取更详细的信息</li>

    <li>查看"模型库"了解不同水果的成熟度特征</li>

</ul>

'''

]

return general_responses[hash(message) % len(general_responses)]

def generate_image_qa_response(question):
    question_lower = question.lower()

    if '成熟度' in question_lower or '成熟' in question_lower:
        return ''

        <p>根据图片分析， 这个水果的成熟度可以通过以下特征判断： </p>

        <ul>

            <li><strong>颜色</strong>： 观察水果的颜色是否达到该品种成熟时的
标准颜色</li>

            <li><strong>质地</strong>： 成熟的水果质地适中， 有弹性</li>

            <li><strong>光泽</strong>： 成熟的水果表面有自然光泽</li>

        </ul>

        <p>建议您使用我们的检测功能， 上传图片后系统会自动给出准确的成熟度判
断。</p>

```

```

'''
elif '储存' in question_lower or '保存' in question_lower:

    return '''

    <p>根据图片中的水果成熟度，储存建议如下： </p>

    <ul>

        <li><strong>成熟水果</strong>： 建议冷藏保存，温度 8-12°C，可保存
7-14 天</li>

        <li><strong>过熟水果</strong>： 建议立即食用或加工，不宜储存</li>

        <li><strong>未成熟水果</strong>： 可以常温催熟，或冷藏延长储存时
间</li>

    </ul>

    <p>具体的储存方法请根据检测结果在"智能推荐"中获取详细建议。</p>
'''

else:

    return '''

    <p>根据图片分析，这个水果看起来很健康! </p>

    <p>我可以帮您： </p>

    <ul>

        <li>判断成熟度</li>

        <li>提供储存建议</li>

        <li>推荐最佳食用时间</li>

        <li>介绍营养价值</li>

    </ul>

    <p>请告诉我您想了解哪方面的信息，我会为您提供更详细的回答。</p>
'''

def generate_recommendations(fruit_type, maturity_level, recommendation_type):

```

```
fruit_info = FRUIT_KNOWLEDGE_BASE.get(fruit_type)

if not fruit_info:

    return '<p>抱歉，暂不支持该水果类型的推荐。</p>'

maturity_info = fruit_info['maturity_levels'].get(maturity_level)

if not maturity_info:

    return '<p>抱歉，暂不支持该成熟度的推荐。</p>'

fruit_name = fruit_info['name']

maturity_name = maturity_info['name']

recommendations = []

if recommendation_type in ['all', 'storage']:

    recommendations.append(f'''

    <h4>    储存建议</h4>

    <p><strong>{fruit_name} ({maturity_name}) 储存方法: </strong></p>

    <p>{maturity_info['storage']}</p>

    ''')

if recommendation_type in ['all', 'harvest']:

    recommendations.append(f'''

    <h4>    采摘建议</h4>

    <p><strong>{fruit_name} ({maturity_name}) 采摘时机: </strong></p>

    <p>{maturity_info['harvest']}</p>

    ''')
```

```

if recommendation_type in ['all', 'market']:

    recommendations.append(f"""

    <h4>    销售建议</h4>

    <p><strong>{fruit_name} ({maturity_name}) 市场策略: </strong></p>

    <p>{maturity_info['market']}</p>

    """)

if recommendation_type in ['all', 'processing']:

    recommendations.append(f"""

    <h4>    加工建议</h4>

    <p><strong>{fruit_name} ({maturity_name}) 加工方式: </strong></p>

    <p>{maturity_info['processing']}</p>

    """)

if recommendation_type == 'all':

    recommendations.append(f"""

    <h4>    综合建议</h4>

    <p><strong>成熟度描述: </strong>{maturity_info['description']}</p>

    <p><strong>种植技巧: </strong></p>

    <ul>{''.join([f'<li>{tip}</li>' for tip in fruit_info['planting_tips']])}</ul>

    """)

return ''.join(recommendations)

```

models.py

```
from flask_sqlalchemy import SQLAlchemy
```

```
from datetime import datetime
```

```
db = SQLAlchemy()
```

```
class User(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    username = db.Column(db.String(50), unique=True, nullable=False)
```

```
    email = db.Column(db.String(100), unique=True, nullable=False)
```

```
    password = db.Column(db.String(100), nullable=False)
```

```
    role = db.Column(db.String(20), nullable=False) # farmer, consumer, business
```

```
    reset_token = db.Column(db.String(100), nullable=True)
```

```
    reset_token_expiry = db.Column(db.DateTime, nullable=True)
```

```
class Detection(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

```
    fruit_type = db.Column(db.String(50), nullable=False) # cucumber, apple, tomato
```

```
    maturity_level = db.Column(db.Integer, nullable=False) # 1-4
```

```
    confidence = db.Column(db.Float, nullable=False)
```

```
    image_path = db.Column(db.String(200), nullable=False)
```

```
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

```
class Dataset(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    filename = db.Column(db.String(200), nullable=False)
```



```
fruit_type = db.Column(db.String(50), nullable=False) # cucumber, apple, tomato
maturity_level = db.Column(db.Integer, nullable=False) # 1-4
image_path = db.Column(db.String(500), nullable=False)
label_path = db.Column(db.String(500), nullable=True)
is_labeled = db.Column(db.Boolean, default=False)
user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
created_at = db.Column(db.DateTime, default=datetime.utcnow)
```