Backend Integration Documentation

Plant Disease Detection API

Table of Contents

Overview

This API provides plant disease detection using Google's Gemini AI. It accepts plant images and returns detailed analysis including plant identification, disease detection, treatment recommendations, and prevention measures in farmer-friendly language.

Key Features

- **AI-Powered Analysis**: Uses Gemini 3 Flash Preview model
- **Farmer-Friendly Output**: Simple, non-technical language
- **Multiple Image Formats**: Supports PNG, JPG, JPEG, GIF, WebP, BMP
- **File Size Limit**: Maximum 16MB per image
- **RESTful API**: JSON responses for easy integration

API Base URL

Development:

```
http://localhost:5000
```

Production:

```
http://your-production-domain.com
```

Authentication

Currently, the API does not require authentication for requests. However, it uses a Gemini API key stored in the `.env` file on the server side.

Environment Setup

Ensure the following environment variable is set on the server:

```
GEMINI_API_KEY=your_gemini_api_key_here
```

Endpoints

1. Home Page (Web Interface)

Endpoint: `/`

Method: `GET`, `POST`

Description: Web interface for uploading and analyzing plant images

Content-Type: `multipart/form-data`

Note: This endpoint returns HTML and is intended for browser use, not API integration.

---

2. Plant Disease Classifier API

Endpoint: `/api/plan-deseases-classifier`

Method: `POST`

Description: Analyze plant images and return disease detection results in JSON format

Content-Type: `multipart/form-data`

Request Parameters

Parameter	Type	Required	Description
`file`	File	Yes	Plant image file (PNG, JPG, JPEG, GIF, WebP, BMP)

Response Format

```
{
  "response": "string - AI analysis result",
  "image_path": "string - Server path to uploaded image",
  "loading": false
}
```

Response Fields

Field	Type	Description
`response`	string	Detailed analysis from Gemini AI including plant identification, disease detection, treatment, and prevention
`image_path`	string	Server file path where the uploaded image is stored (e.g., `static/uploads/uuid_filename.jpg`)
`loading`	boolean	Always `false` in the response (indicates processing is complete)

Request/Response Examples

Example 1: Successful Disease Detection

Request (cURL):

```
curl -X POST http://localhost:5000/api/plan-deseases-classifier \
  -F "file=@/path/to/plant_image.jpg"
```

Request (JavaScript/Fetch):

```
const formData = new FormData();
formData.append('file', fileInput.files[0]);

fetch('http://localhost:5000/api/plan-deseases-classifier', {
```

```
  method: 'POST',
  body: formData
})
.then(response => response.json())
.then(data => {
  console.log('Analysis:', data.response);
  console.log('Image Path:', data.image_path);
})
.catch(error => console.error('Error:', error));
```

Response (200 OK):

```
{
  "response": "Plant: Tomato\n\nDisease Detected: Early Blight\n\nExplana
  "image_path": "static/uploads/a1b2c3d4-e5f6-7890-abcd-ef1234567890_toma
  "loading": false
}
```

Example 2: No File Uploaded

Request:

```
curl -X POST http://localhost:5000/api/plan-deseases-classifier
```

Response (302 Redirect):
The API will redirect back to the same URL with a flash message. This is not ideal for API usage and should be handled by checking for the file before making the request.

Example 3: Invalid File Type

Request:

```
curl -X POST http://localhost:5000/api/plan-deseases-classifier \
  -F "file=@document.pdf"
```

Response (302 Redirect):
The API will redirect if the file type is not allowed. Ensure you only send image files with allowed extensions.

Example 4: Gemini API Error

Response (200 OK with error message):

```
{
  "response": "Error: 429 - Quota exceeded",
  "image_path": "static/uploads/a1b2c3d4-e5f6-7890-abcd-ef1234567890_plar
  "loading": false
}
```

Error Handling

Common Error Scenarios

1. Missing File

- **Cause:** No file included in the request
- **Behavior:** Redirects with flash message "No file part"
- **Solution:** Always include a file in the `file` field

2. Empty Filename

- **Cause:** File field is present but no file selected
- **Behavior:** Redirects with flash message "No selected file"
- **Solution:** Ensure a file is actually selected before submission

3. Invalid File Type

- **Cause:** File extension not in allowed list
- **Behavior:** Request is redirected without processing
- **Solution:** Only send files with extensions: png, jpg, jpeg, gif, webp, bmp

4. File Too Large

- **Cause:** File exceeds 16MB limit
- **Behavior:** Flask returns 413 Request Entity Too Large
- **Solution:** Compress or resize images before upload

5. Gemini API Errors

- **Cause:** API key issues, quota exceeded, network problems
- **Behavior:** Returns error message in `response` field
- **Example:** `"Error: 401 - Invalid API key"`

- **Solution:** Check API key, quota, and network connectivity

---

Integration Examples

PHP/Laravel Integration

Option 1: Using Laravel HTTP Client

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Http;

class PlantDiseaseController extends Controller
{
    public function analyzePlant(Request $request)
    {
        // Validate the uploaded file
        $request->validate([
            'plant_image' => 'required|image|mimes:png,jpg,jpeg,gif,webp,
        ]);

        // Get the uploaded file
        $file = $request->file('plant_image');

        // Send to Python API
        $response = Http::attach(
            'file',
            file_get_contents($file->getRealPath()),
            $file->getClientOriginalName()
        )->post('http://localhost:5000/api/plan-deseases-classifier');

        // Check if request was successful
        if ($response->successful()) {
            $result = $response->json();

            return response()->json([
                'success' => true,
                'analysis' => $result['response'],
```

```
                'image_path' => $result['image_path']
            ]);
        }

        return response()->json([
            'success' => false,
            'message' => 'Failed to analyze image'
        ], 500);
    }
}
```

Option 2: Using cURL

```
<?php

function analyzePlantDisease($imagePath)
{
    $apiUrl = 'http://localhost:5000/api/plan-deseases-classifier';

    $cFile = new \CURLFile($imagePath, mime_content_type($imagePath), bas

    $postData = ['file' => $cFile];

    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $apiUrl);
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $postData);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    $response = curl_exec($ch);
    $httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
    curl_close($ch);

    if ($httpCode == 200) {
        return json_decode($response, true);
    }

    return null;
}

// Usage
$result = analyzePlantDisease('/path/to/plant_image.jpg');
if ($result) {
```

```
    echo "Analysis: " . $result['response'];
}
```

Node.js/Express Integration

```
const express = require('express');
const multer = require('multer');
const FormData = require('form-data');
const axios = require('axios');
const fs = require('fs');

const app = express();
const upload = multer({ dest: 'uploads/' });

app.post('/analyze', upload.single('plant_image'), async (req, res) => {
  try {
    const formData = new FormData();
    formData.append('file', fs.createReadStream(req.file.path));

    const response = await axios.post(
      'http://localhost:5000/api/plan-deseases-classifier',
      formData,
      {
        headers: formData.getHeaders()
      }
    );

    // Clean up uploaded file
    fs.unlinkSync(req.file.path);

    res.json({
      success: true,
      analysis: response.data.response,
      imagePath: response.data.image_path
    });
  } catch (error) {
    res.status(500).json({
      success: false,
      message: error.message
    });
  }
});
```

```
app.listen(3000, () => console.log('Server running on port 3000'));
```

Python/Django Integration

```
import requests
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def analyze_plant(request):
    if request.method == 'POST' and request.FILES.get('plant_image'):
        image_file = request.FILES['plant_image']

        # Prepare the file for upload
        files = {'file': (image_file.name, image_file.read(), image_file.

        # Send to Flask API
        response = requests.post(
            'http://localhost:5000/api/plan-deseases-classifier',
            files=files
        )

        if response.status_code == 200:
            result = response.json()
            return JsonResponse({
                'success': True,
                'analysis': result['response'],
                'image_path': result['image_path']
            })

        return JsonResponse({
            'success': False,
            'message': 'Failed to analyze image'
        }, status=500)

    return JsonResponse({'error': 'No image provided'}, status=400)
```

Best Practices

1. File Validation

Always validate files on your backend before sending to the API:

```
// Laravel example
$request->validate([
    'plant_image' => 'required|image|mimes:png,jpg,jpeg,gif,webp,bmp|max:
]);
```

2. Error Handling

Implement comprehensive error handling:

```
try {
    $response = Http::attach(...)->post(...);

    if ($response->successful()) {
        // Process success
    } else {
        // Handle HTTP errors
        Log::error('API Error: ' . $response->status());
    }
} catch (\Exception $e) {
    // Handle network/connection errors
    Log::error('Connection Error: ' . $e->getMessage());
}
```

3. Timeout Configuration

Set appropriate timeouts for API calls (Gemini AI can take 5-15 seconds):

```
$response = Http::timeout(30)->attach(...)->post(...);
```

4. Async Processing (Recommended)

For production, consider using job queues for image analysis:

```
// Laravel Job
dispatch(new AnalyzePlantImageJob($imagePath));
```

5. Response Caching

Cache analysis results to avoid redundant API calls:

```
$cacheKey = 'plant_analysis_' . md5_file($imagePath);
$result = Cache::remember($cacheKey, 3600, function() use ($imagePath) {
    return $this->callAnalysisAPI($imagePath);
});
```

6. Image Optimization

Resize/compress images before sending to reduce processing time:

```
// Resize to max 1024x1024 before sending
$image = Image::make($file)->resize(1024, 1024, function ($constraint) {
    $constraint->aspectRatio();
    $constraint->upsize();
});
```

7. Rate Limiting

Implement rate limiting to prevent API abuse:

```
// Laravel middleware
Route::post('/analyze')->middleware('throttle:10,1');
```

8. Monitoring

Log all API calls for monitoring and debugging:

```
Log::info('Plant analysis requested', [
    'user_id' => auth()->id(),
    'filename' => $file->getClientOriginalName(),
    'size' => $file->getSize()
]);
```

Production Deployment Considerations

1. Environment Configuration

```
# .env file
PLANT_DISEASE_API_URL=http://your-api-server:5000
GEMINI_API_KEY=your_production_api_key
```

2. HTTPS/SSL

Always use HTTPS in production:

```
$apiUrl = env('PLANT_DISEASE_API_URL', 'https://api.yourdomain.com');
```

3. Load Balancing

For high traffic, deploy multiple API instances behind a load balancer.

4. Health Checks

Implement health check endpoint monitoring:

```
// Check if API is available
$health = Http::get('http://localhost:5000/api/health');
```

5. Fallback Mechanism

Implement fallback when API is unavailable:

```
if (!$response->successful()) {
    // Queue for retry or use backup service
    dispatch(new RetryAnalysisJob($imagePath))->delay(now()->addMinutes(5
}
```

Support & Troubleshooting

Common Issues

Issue: API returns empty response

- **Solution:** Check if Flask server is running and accessible

Issue: File upload fails

- **Solution:** Verify file size < 16MB and format is supported

Issue: Slow response times

- **Solution:** Gemini API can take 5-15 seconds; implement async processing

Issue: "Error: 429 - Quota exceeded"

- **Solution:** Check Gemini API quota limits and upgrade if needed

Issue: CORS errors (if calling from browser)

- **Solution:** Ensure Flask-CORS is configured properly in the API

API Versioning

Current Version: **v1**

Future versions will be accessible via:

`/api/v2/plan-deseases-classifier`

Contact & Support

For technical support or questions about integration:

- **Email:** support@yourcompany.com
- **Documentation:** https://docs.yourcompany.com
- **Issue Tracker:** https://github.com/yourcompany/plant-disease-api/issues

---

Last Updated: January 26, 2026
API Version: 1.0.0
Documentation Version: 1.0.0