CARFAX Integration Documentation

Overview

This document describes the integration of CARFAX-Wrapper functionality into the auction automation system. The integration is based on the amattu2/CARFAX-Wrapper (https://github.com/amattu2/CARFAX-Wrapper) PHP library, adapted for Python.

What Was Implemented

1. CarfaxServiceHistory Class

A Python implementation of the PHP wrapper's ServiceHistory functionality:

```
from integrations.carfax import CarfaxServiceHistory

# Configure credentials
CarfaxServiceHistory.set_product_data_id("your_16_char_id")
CarfaxServiceHistory.set_location_id("your_location_id")

# Fetch vehicle history
history = CarfaxServiceHistory.get("1G1GCCBX3JX001788")
```

Features:
- Static class methods matching PHP wrapper API
- VIN validation (17 alphanumeric characters)
- Credential validation (16-character product_data_id, 1-50 character location_id)
- Structured response format identical to PHP wrapper
- Comprehensive error handling

2. Enhanced CarfaxIntegrator Class

The existing `CarfaxIntegrator` class has been enhanced to support multiple data sources:

1. **Primary**: CARFAX Service History API (via wrapper)
2. **Secondary**: Legacy CARFAX API (if available)
3. **Fallback**: Web scraping

3. Response Format

The wrapper returns data in the same format as the PHP version:

```json
{
  "Decode": {
    "VIN": "1G1GCCBX4JX001298",
    "Year": "2011",
    "Make": "CADILLAC",
    "Model": "LUXURY",
    "Trim": "",
    "Driveline": ""
  },
  "Overview": [
    {
      "Name": "Tire rotation",
      "Date": "12/24/2013",
      "Odometer": 42185
    }
  ],
  "Records": [
    {
      "Date": "01/12/2011",
      "Odometer": 5,
      "Services": ["Vehicle serviced", "Pre-delivery inspection completed"],
      "Type": "Service"
    }
  ]
}
```


Configuration

Environment Variables

Add these to your `.env` file:

```
# CARFAX Service History API (preferred method)
CARFAX_PRODUCT_DATA_ID=your_16_character_product_data_id
CARFAX_LOCATION_ID=your_location_id

# Legacy API support (optional)
CARFAX_API_KEY=your_legacy_api_key
```

Configuration File

The `config/config.yaml` has been updated:

```yaml
integrations:
  carfax:
    enabled: true
    # Legacy API key support
    api_key: "${CARFAX_API_KEY}"
    # CARFAX Service History API credentials (preferred method)
    product_data_id: "${CARFAX_PRODUCT_DATA_ID}"
    location_id: "${CARFAX_LOCATION_ID}"
    # Configuration options
    use_wrapper_api: true
    fallback_scraping: true
```

API Requirements

CARFAX Service Data Transfer Facilitation Agreement

To use the CARFAX Service History API, you need:

1. **Business Agreement**: A CARFAX Service Data Transfer Facilitation Agreement
2. **Credentials**:
   - Product Data ID (16 characters) - acts as API key
   - Location ID (1-50 characters) - identifies your location
3. **API Access**: Access to `https://servicesocket.carfax.com/data/1`

How to Obtain Credentials

1. Contact CARFAX Business Development Team
2. Establish a Service Data Transfer Facilitation Agreement
3. Receive your Product Data ID and Location ID during account setup
4. Configure the credentials in your environment

Usage Examples

Basic Usage

```python
from integrations.carfax import CarfaxIntegrator

# Initialize integrator
integrator = CarfaxIntegrator()

# Get vehicle history
vin = "1G1GCCBX3JX001788"
history = integrator.get_vehicle_history(vin)

# Analyze for red flags
analysis = integrator.analyze_history_flags(history)

print(f"Overall risk: {analysis['overall_risk']}")
print(f"Red flags: {analysis['red_flags']}")
```

Direct Wrapper Usage

```python
from integrations.carfax import CarfaxServiceHistory

# Configure once
CarfaxServiceHistory.set_product_data_id(os.getenv('CARFAX_PRODUCT_DATA_ID'))
CarfaxServiceHistory.set_location_id(os.getenv('CARFAX_LOCATION_ID'))

# Use multiple times
for vin in vehicle_vins:
    try:
        history = CarfaxServiceHistory.get(vin)
        print(f"Vehicle: {history['Decode']['Year']} {history['Decode']['Make']}")
        print(f"Service records: {len(history['Records'])}")
    except Exception as e:
        print(f"Error for VIN {vin}: {e}")
```

Testing

Run Tests

```
# Simple test (no browser dependencies)
python test_carfax_simple.py

# Full integration test (requires browser setup)
python test_carfax_integration.py
```

Test Results

The integration has been tested and verified:

- ✅ VIN validation working
- ✅ Credential validation working
- ✅ API endpoint reachable
- ✅ Request format correct
- ✅ Response structure properly formatted
- ✅ Error handling working

Error Handling

Common Errors

1. **Invalid VIN**: `ValueError: VIN must be exactly 17 alphanumeric characters`
2. **Missing Credentials**: `RuntimeError: Product Data ID must be set before making requests`
3. **API Access**: `CARFAX API error: User does not have access to this Product`
4. **Network Issues**: `CARFAX API request failed: Connection timeout`

Fallback Behavior

If the wrapper API fails, the system automatically falls back to:
1. Legacy CARFAX API (if configured)
2. Web scraping (if enabled)
3. Empty result (graceful degradation)

Integration Benefits

Advantages Over Direct API Calls

1. **Structured Interface**: Clean, object-oriented API
2. **Validation**: Built-in VIN and credential validation
3. **Error Handling**: Comprehensive error management
4. **Consistency**: Same interface as established PHP wrapper
5. **Fallback Support**: Multiple data source options
6. **Type Safety**: Full type hints and documentation

Backward Compatibility

- Existing code using `CarfaxIntegrator` continues to work
- Legacy API key configuration still supported

- Same analysis methods and response formats
- No breaking changes to existing functionality

Security Considerations

Credential Management

- Store credentials in environment variables
- Never commit credentials to version control
- Use different credentials for development/production
- Rotate credentials regularly

API Usage

- Respect rate limits (10 requests/minute default)
- Implement proper retry logic
- Log API usage for monitoring
- Handle sensitive data appropriately

Troubleshooting

Common Issues

1. **“User does not have access to this Product”**
   - Verify your CARFAX agreement covers Service History API
   - Check that Product Data ID is correct
   - Ensure Location ID is properly configured

2. **“Product Data ID must be exactly 16 characters”**
   - Verify the Product Data ID length
   - Check for extra spaces or characters
   - Confirm you’re using the correct credential

3. **“VIN must be exactly 17 alphanumeric characters”**
   - Ensure VIN is properly formatted
   - Remove any spaces or special characters
   - Verify VIN is valid and complete

Debug Mode

Enable debug logging to troubleshoot issues:

```
import logging
logging.basicConfig(level=logging.DEBUG)

# Your CARFAX integration code here
```

Future Enhancements

Planned Features

1. **QuickVIN Integration**: Add license plate to VIN decoding
2. **FTP Reporting**: Implement service history reporting to CARFAX

3. **Batch Processing**: Support for multiple VIN lookups
4. **Caching**: Add response caching to reduce API calls
5. **Metrics**: Enhanced usage tracking and analytics

Contributing

To contribute to the CARFAX integration:

1. Follow the existing code patterns
2. Add comprehensive tests
3. Update documentation
4. Ensure backward compatibility
5. Test with real CARFAX credentials when possible

Support

For issues related to:

- **Integration Code**: Create GitHub issue in this repository
- **CARFAX API Access**: Contact CARFAX Business Development
- **Credentials**: Contact your CARFAX account representative
- **PHP Wrapper**: See amattu2/CARFAX-Wrapper (https://github.com/amattu2/CARFAX-Wrapper)

License

This integration maintains compatibility with the original CARFAX-Wrapper license (AGPL-3.0) while being part of the auction automation system.