Warranty Wallet

Warranty Wallet 🛡️

An intelligent warranty and claims management system that connects customers, retailers, and administrators through a unified platform. Built with Django REST Framework, it features AI-powered claim analysis, automated priority detection, and comprehensive warranty tracking.

version 1.0.0

python 3.12

django 5.2.7

license MIT

📋 Table of Contents

🎯 Overview

Warranty Wallet revolutionizes warranty management by providing:

- **Digital Warranty Storage** — Never lose a warranty again
- **AI-Powered Claims** — Automatic priority detection based on description severity
- **Multi-Platform Support** — Web dashboard + Mobile API
- **Real-Time Analytics** — AI-driven insights for retailers
- **Automated Notifications** — Keep all parties informed
- **Smart Document Management** — Upload receipts, warranties, and claim attachments

Problem Statement

Traditional warranty management is:

- ❌ Paper-based and easy to lose
- ❌ Manual claim processing is slow
- ❌ No visibility into claim status
- ❌ Difficult to track warranty expiration
- ❌ Poor communication between customers and retailers

Our Solution

Warranty Wallet provides:

- ✅ Digital warranty storage with automatic reminders
- ✅ AI-powered claim triaging (High/Medium/Low priority)

- ✅ Real-time claim tracking and notifications
- ✅ Automated warranty expiration alerts
- ✅ Seamless communication between all parties
- ✅ Analytics dashboard for retailers

🚀 Key Features

1. Smart Warranty Management

- 📄 **Receipt Upload & OCR** — Upload receipts and automatically extract warranty information
- 🔔 **Expiration Reminders** — Get notified before warranties expire
- 📱 **Mobile & Web Access** — Access warranties from anywhere
- 🏪 **Store Integration** — Link warranties to specific stores and retailers
- 📦 **Product Specifications** — Track model, serial number, IMEI, storage, color

2. AI-Powered Claims Processing

- 🤖 **Automatic Priority Detection** — AI analyzes claim descriptions to set priority:
  - **High Priority**: Fire, explosion, safety hazards, complete failure
  - **Medium Priority**: Malfunctions, performance issues, battery problems
  - **Low Priority**: Cosmetic damage, minor issues
- 📊 **Claim Analytics** — AI-powered insights on:
  - Most claimed products
  - Slowest processing claims
  - Claim reasons analysis
  - Rejection pattern detection
- 🎯 **Smart Recommendations** — AI suggests preventive measures based on claim patterns

3. Customer-Uploaded Warranties

- 📸 **Manual Upload** — Customers can upload warranty documents independently
- 🖼️ **Image Support** — Upload photos or scans of warranty cards
- 📅 **Simple Tracking** — Just product name, expiry date, and image
- 🔍 **Status Monitoring** — Active, Expiring Soon, Expired

4. Comprehensive Claims System

- 📝 **Easy Claim Submission** — Submit claims with issue description and attachments
- 📎 **Multi-Attachment Support** — Upload up to 10 files per claim (max 10MB each)
- 💬 **Internal Notes** — Retailer/admin collaboration notes
- 📈 **Status Tracking** — In Review → Approved/Rejected
- 💰 **Cost Estimation** — Track estimated vs actual costs
- 📊 **Export to CSV** — Download individual or bulk claims

5. Real-Time Notifications

- 🔔 **New Claim Alerts** — Retailers notified immediately
- 📢 **Status Updates** — Customers notified of claim progress
- 💬 **Note Notifications** — Alerts when notes or attachments are added
- ⏰ **Warranty Expiration** — Reminders before expiry

6. Advanced Analytics Dashboard

- 📊 **Overview Statistics**:
  - Total receipts, warranties, claims
  - Approval/rejection rates
  - Month-over-month trends

    - Average response time
- 🎯 **AI Insights** (for retailers):
    - Top claimed products with approval rates
    - Slow processing identification
    - Claim reason categorization
    - Rejection reason analysis
    - Actionable recommendations
- 📈 **Time-Based Filtering** — 1 month, 3 months, 6 months, 1 year
- 📩 **Data Export** — Export analytics to CSV

7. Multi-Role Access Control

- 👤 **Customer Portal**:
    - Upload receipts and warranties
    - File claims
    - Track claim status
    - View warranty history
    - Upload claim attachments
- 🏪 **Retailer Dashboard**:
    - Manage claims
    - Add internal notes
    - Update claim status and priority
    - View analytics
    - Manage store information
- 👨‍💼 **Admin Panel**:
    - Full system access
    - User management

    - Global analytics
    - System configuration

8. Document Management

- 📄 **Receipt Storage** — Organized by date with automatic folder structure
- 🖼️ **Warranty Images** — Store warranty documents and cards
- 📎 **Claim Attachments** — Upload evidence photos and documents
- 🏪 **Store Logos** — Brand identity management
- 📱 **Media Optimization** — Automatic file organization by year/month

9. RESTful API

- 🔐 **JWT Authentication** — Secure token-based auth with refresh tokens
- 📱 **Mobile-Friendly** — Dedicated customer endpoints
- 📚 **Auto-Generated Docs** — Swagger/ReDoc documentation
- 🔄 **CRUD Operations** — Full create, read, update, delete support
- 🎯 **Filtering & Search** — Advanced query capabilities

10. Performance & Optimization

- ⚡ **Query Optimization** — N+1 query prevention with select_related/prefetch_related
- 📊 **Subquery Aggregation** — Efficient complex queries
- 🚀 **Conditional Aggregation** — Optimized statistics calculation
- 🔍 **Database Indexing** — Fast lookups on critical fields
- 📈 **Query Counter** — Development tool for optimization

💻 Technology Stack

Backend

- **Django 5.2.7** — Web framework

- **Django REST Framework 3.15.2** — API development
- **PostgreSQL** — Production database
- **SQLite** — Development database
- **Python 3.12** — Programming language

AI & Machine Learning

- **Hugging Face Inference API** — AI model hosting
- **facebook/bart-large-mnli** — Zero-shot text classification
- **facebook/bart-large-cnn** — Text summarization
- **huggingface-hub 0.20.3** — API client

Authentication & Security

- **djangorestframework-simplejwt 5.3.1** — JWT authentication
- **Django CORS Headers 4.6.0** — Cross-origin resource sharing

API Documentation

- **drf-yasg 1.21.7** — Swagger/OpenAPI documentation
- **Swagger UI** — Interactive API testing
- **ReDoc** — Alternative API documentation

Additional Libraries

- **Pillow 11.0.0** — Image processing
- **django-filter 24.3** — Advanced filtering
- **python-decouple 3.8** — Environment configuration
- **python-dateutil 2.9.0** — Date manipulation
- **psycopg2-binary 2.9.10** — PostgreSQL adapter

Development Tools

- **django-query-counter 0.4.0** — Query optimization monitoring

- **Git** — Version control
- **GitHub** — Code hosting

🏗️ Architecture

System Architecture

```
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│                                   Warranty Wallet                                            │
├──────────────────────────────────────────────────────────────────────────────────────────────┤
│                                                                                              │
│  ┌──────────────────────┐  ┌──────────────────────┐  ┌──────────────────────┐                │
│  │   Customer   │  │   Retailer   │  │    Admin     │      │
│  │    Portal    │  │  Dashboard  │  │    Panel     │      │
│  └──────────┬───────────┘  └──────────┬───────────┘  └──────────┬───────────┘                │
│                                                                                              │
│       │            │            │           │                                                │
│       └────────────────────────┴────────────────────────┘           │
│                    │                       │                                                 │
│  ┌─────────────────────────────────▼─────────────────────────────────────────────┐      │
│  │            RESTful API Layer                 │      │
│  │    (JWT Auth, Permissions, Serialization)        │      │
│  └──────────────────────────────────────┬──────────────────────────────────────────────┘      │
│                    │                       │                                                 │
│  ┌─────────────────────────────────▼─────────────────────────────────────────────┐      │
│  │            Business Logic Layer              │      │
│  ├──────────────────────┬──────────────────────┬──────────────────────┬──────────────┤      │
│  │   Accounts   │  Receipts   │  Warranties  │ Claims │      │
```

```
│  │  Analytics  │    Store    │Notifications │       │      │
│  └──────────────────────┴──────────────────────┴──────────────────────┴──────
──────┘      │
│                    │                       │
│  ┌──────────────────────────────────────────▼──────────────────────────────
────────┐      │
│  │              AI Services Layer                │      │
│  ├──────────────────────────────┬──────────────────────────────────────────
────────┤      │
│  │ Priority Detection  │  Analytics AI Insights      │      │
│  │ (Claim Severity)    │  (Pattern Recognition)      │      │
│  └──────────────────────────────┴──────────────────────────────────────────
────────┘      │
│                    │                       │
│  ┌──────────────────────────────────────────▼──────────────────────────────
────────┐      │
│  │            Data Persistence Layer              │      │
│  │         (PostgreSQL / SQLite Database)         │      │
│  └──────────────────────────────────────────────────────────────────────────
────────┘      │
│                                       │
└──────────────────────────────────────────────────────────────────────────────
──────────────┘

External Services:
┌──────────────────────────┐
│  Hugging Face API    │  ← AI Models
│  (Inference Endpoint) │
└──────────────────────────┘
```

Application Structure

```
Warranty-Wallet/
├── accounts/        # User management & authentication
├── analytics/       # AI-powered analytics & insights
```

```
├── claims/          # Claims processing & AI priority
├── config/          # Django settings & configuration
├── notifications/   # Real-time notification system
├── receipts/        # Receipt & item management
├── store/           # Store/retailer information
├── warranties/      # Warranty & customer warranty management
├── media/           # User-uploaded files
├── static/          # Static assets
└── venv/            # Virtual environment
```

🤖 AI Features

1. Automatic Claim Priority Detection

How It Works:

1. Customer submits claim with description
2. AI analyzes text for:
   - Urgency indicators (fire, explosion, emergency)
   - Severity keywords (broken, unusable, dangerous)
   - Safety concerns (smoke, sparking, injury)
3. AI classifies into priority levels
4. Priority is automatically set (can be manually overridden)

Example:

```
Description: "Phone caught fire while charging!"
AI Detection: HIGH PRIORITY
Reason: Safety hazard detected

Description: "Battery drains faster than normal"
AI Detection: MEDIUM PRIORITY
Reason: Functional issue
```

```
Description: "Small scratch on back cover"
AI Detection: LOW PRIORITY
Reason: Cosmetic damage
```

Models Used:

- **facebook/bart-large-mnli** — Zero-shot classification
- Runs on Hugging Face Inference API (no local compute needed)

Fallback System:

- If AI unavailable → Keyword-based detection
- Always functional, never fails

2. AI-Powered Analytics Insights

For Retailers:

1. **Top Claimed Products Analysis** — Identifies products with most claims, calculates approval rates, highlights quality issues
2. **Slow Processing Detection** — Identifies claims taking too long, calculates average processing time, suggests workflow improvements
3. **Claim Reason Classification** — Categories: Screen damage, Battery issue, Hardware failure, etc. Uses AI for pattern recognition and provides actionable insights
4. **Rejection Reason Analysis** — Identifies common rejection patterns, helps improve customer education, reduces future claim rejections
5. **AI Summary Generation** — Summarizes key analytics findings, provides concise insights, uses facebook/bart-large-cnn model

API Endpoint:

```
GET /api/analytics/ai-insights/?period=6months
```

👥 User Roles

Customer

Capabilities:

- ✅ Register and manage account
- ✅ Upload receipts with automatic parsing
- ✅ Upload manual warranty documents
- ✅ View all warranties (receipt-based + manual)
- ✅ File claims with automatic priority detection
- ✅ Upload claim attachments (photos/documents)
- ✅ Track claim status in real-time
- ✅ Receive notifications for claim updates
- ✅ View warranty expiration dates
- ✅ Access customer-friendly mobile endpoints

Retailer

Capabilities:

- ✅ All customer capabilities
- ✅ Manage claims for their store
- ✅ Add internal notes to claims
- ✅ Update claim status (Approve/Reject)
- ✅ Change claim priority
- ✅ Add estimated and actual costs
- ✅ View AI-powered analytics dashboard
- ✅ Export claims to CSV
- ✅ Manage store information
- ✅ View customer claim history
- ✅ Receive notifications for new claims

Admin

Capabilities:

- ✅ All retailer capabilities
- ✅ Access to all system data
- ✅ User management (create/edit users)
- ✅ Global analytics across all stores
- ✅ System configuration
- ✅ Database management via Django Admin
- ✅ Full CRUD on all models

📚 API Documentation

Authentication Endpoints

```
POST /api/accounts/register/
POST /api/accounts/login/
POST /api/accounts/token/refresh/
GET  /api/accounts/profile/
PUT  /api/accounts/profile/
```

Receipt Endpoints

```
GET    /api/receipts/              # List receipts
POST   /api/receipts/              # Create receipt with items
GET    /api/receipts/{id}/         # Receipt details
PUT    /api/receipts/{id}/         # Update receipt
DELETE /api/receipts/{id}/         # Delete receipt
GET    /api/receipts/export/       # Export to CSV
```

Warranty Endpoints

```
GET    /api/warranties/                # List warranties (receipt-based)
POST   /api/warranties/                # Create warranty
GET    /api/warranties/{id}/           # Warranty details

# Customer-uploaded warranties
GET    /api/warranties/customer/       # List customer warranties
POST   /api/warranties/customer/       # Upload warranty
GET    /api/warranties/customer/{id}/  # Details
PUT    /api/warranties/customer/{id}/  # Update
DELETE /api/warranties/customer/{id}/  # Delete
```

Claims Endpoints

```
# General claims (all roles)
GET    /api/claims/                    # List claims
POST   /api/claims/                    # Create claim
GET    /api/claims/{id}/               # Claim details
PATCH  /api/claims/{id}/               # Update claim (retailers only)
POST   /api/claims/{id}/add_note/      # Add internal note
POST   /api/claims/{id}/upload_attachment/ # Upload file
GET    /api/claims/{id}/attachments/   # List attachments
GET    /api/claims/{id}/download/      # Download as CSV
GET    /api/claims/export/             # Export all claims

# Customer-specific endpoints
GET    /api/claims/my/                 # My claims
POST   /api/claims/my/create/          # Create claim
GET    /api/claims/my/{id}/            # Claim details
POST   /api/claims/my/{id}/add-attachment/ # Add attachment
```

Analytics Endpoints

```
GET /api/analytics/                    # Overview statistics
GET /api/analytics/ai-insights/        # AI-powered insights
```

```
?period=1month|3months|6months|1year
```

Store Endpoints

```
GET    /api/stores/                    # List stores
POST   /api/stores/                    # Create store
GET    /api/stores/{id}/               # Store details
PUT    /api/stores/{id}/               # Update store
```

Notification Endpoints

```
GET    /api/notifications/             # List notifications
GET    /api/notifications/unread/      # Unread count
POST   /api/notifications/{id}/mark-read/  # Mark as read
POST   /api/notifications/mark-all-read/   # Mark all read
```

Interactive API Documentation

- **Swagger UI:** http://localhost:8000/
- **ReDoc:** http://localhost:8000/redoc/
- **JSON Schema:** http://localhost:8000/swagger.json

🛠️ Installation

Prerequisites

- Python 3.12+
- PostgreSQL 14+ (for production)
- Git
- Virtual environment tool (venv)

Step 1: Clone Repository

```
git clone https://github.com/rakhmatov1337/Warranty-Wallet.gitcd Warranty-Wallet
```

Step 2: Create Virtual Environment

```
# Windows
python -m venv venv
venv\Scripts\activate

# Linux/Mac
python3 -m venv venv
source venv/bin/activate
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
```

Step 4: Environment Configuration

Create a `.env` file in the project root:

```
# Database (Development - SQLite)
DATABASE_URL=sqlite:///db.sqlite3

# Database (Production - PostgreSQL)
# DATABASE_URL=postgresql://user:password@localhost:5432/warranty_wallet

# Django Settings
SECRET_KEY=your-secret-key-here
DEBUG=True
ALLOWED_HOSTS=localhost,127.0.0.1

# AI Configuration
```

```
HF_TOKEN=your-huggingface-token-here

# CORS Settings
CORS_ALLOWED_ORIGINS=http://localhost:3000,http://localhost:5173

# Email Configuration (optional)
EMAIL_BACKEND=django.core.mail.backends.console.EmailBackend
```

Step 5: Database Setup

```
# Run migrations
python manage.py migrate

# Create superuser
python manage.py createsuperuser

# (Optional) Load sample data
python manage.py create_sample_users
python manage.py populate_mock_data
```

Step 6: Collect Static Files

```
python manage.py collectstatic --noinput
```

Step 7: Run Development Server

```
python manage.py runserver
```

Visit `http://localhost:8000/` for Swagger UI documentation.

⚙️ Configuration

Hugging Face AI Setup

1. **Create Account**: https://huggingface.co/2
2. **Get API Token**: Settings → Access Tokens → Create New Token
3. **Add to Environment**:

```
export HF_TOKEN=hf_your_token_here
```

PostgreSQL Production Setup

1. **Install PostgreSQL**:

```
# Ubuntu/Debian
sudo apt install postgresql postgresql-contrib

# macOS
brew install postgresql
```

2. **Create Database**:

```
sudo -u postgres psql
CREATE DATABASE warranty_wallet;
CREATE USER warranty_user WITH PASSWORD 'your_password';
GRANT ALL PRIVILEGES ON DATABASE warranty_wallet TO warranty_use
r;
\q
```

3. **Update .env**:

```
DATABASE_URL=postgresql://warranty_user:your_password@localhost:54
32/warranty_wallet
```

CORS Configuration

For frontend development, update `config/settings.py`:

```
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",  # React
    "http://localhost:5173",  # Vite
    "http://localhost:8080",  # Vue
]
```

📖 Usage Examples

Example 1: Customer Registers and Files Claim

```
import requests

BASE_URL = "http://localhost:8000/api"

# 1. Register
response = requests.post(f"{BASE_URL}/accounts/register/", json={
    "email": "customer@example.com",
    "password": "securepass123",
    "full_name": "John Doe",
    "phone_number": "+1234567890",
    "role": "customer"
})

tokens = response.json()
access_token = tokens['access']

# 2. Upload Receipt
headers = {"Authorization": f"Bearer {access_token}"}
receipt_data = {
    "receipt_number": "RCP-001",
    "store": 1,
    "date": "2025-01-15",
    "total": 999.99,
    "payment_method": "Credit Card",
```

```
    "items": [{
        "product_name": "iPhone 15 Pro Max",
        "model": "A3108",
        "serial_number": "ABC123456",
        "price": 999.99,
        "quantity": 1
    }]
}

receipt = requests.post(
    f"{BASE_URL}/receipts/",
    json=receipt_data,
    headers=headers
).json()

# 3. Create Warranty
warranty_data = {
    "receipt_item_id": receipt['items'][0]['id'],
    "coverage_period_months": 24,
    "provider": "Apple Inc.",
    "coverage_terms": "Full hardware coverage"
}

warranty = requests.post(
    f"{BASE_URL}/warranties/",
    json=warranty_data,
    headers=headers
).json()

# 4. File Claim (AI automatically detects priority)
claim_data = {
    "warranty_id": warranty['id'],
    "issue_summary": "Battery drains extremely fast",
    "detailed_description": "The battery goes from 100% to 0% in just 2 hours
with normal use. Started after 3 months.",
    "category": "Malfunction"
```

```
}

claim = requests.post(
    f"{BASE_URL}/claims/my/create/",
    json=claim_data,
    headers=headers
).json()

print(f"Claim created: {claim['claim_number']}")
print(f"AI-detected priority: {claim['priority']}")  # Output: "Medium"
```

Example 2: Retailer Views Analytics

```
# Login as retailer
response = requests.post(f"{BASE_URL}/accounts/login/", json={
    "email": "retailer@example.com",
    "password": "password123"
})
retailer_token = response.json()['access']
headers = {"Authorization": f"Bearer {retailer_token}"}

# Get AI insights
analytics = requests.get(
    f"{BASE_URL}/analytics/ai-insights/?period=6months",
    headers=headers
).json()

print(f"Total claims: {analytics['overall_statistics']['total_claims']}")
print(f"Approval rate: {analytics['overall_statistics']['approval_rate']}%")
print(f"\nTop claimed product: {analytics['top_claimed_products'][0]['product
_name']}")
print(f"AI Summary: {analytics['ai_summary']}")
```

Example 3: Upload Custom Warranty

```
# Customer uploads warranty document
headers = {"Authorization": f"Bearer {access_token}"}

# Upload warranty image
with open('warranty_card.jpg', 'rb') as f:
    files = {'warranty_image': f}
    data = {
        'product_name': 'Samsung TV 55"',
        'expiry_date': '2027-12-31',
        'notes': 'Extended warranty from Samsung'
    }
    warranty = requests.post(
        f"{BASE_URL}/warranties/customer/",
        data=data,
        files=files,
        headers=headers
    ).json()

print(f"Warranty uploaded: {warranty['id']}")
print(f"Status: {warranty['status']}")  # Active/Expiring Soon/Expired
print(f"Days remaining: {warranty['days_remaining']}")
```

🗄️ Database Schema

Core Models

User (CustomUser)

- Email-based authentication
- Roles: customer, retailer, admin
- Profile information

Store

- Store name, address, contact

- Logo image
- Linked to retailer

Receipt

- Receipt number, date, total
- Linked to customer, retailer, store
- Multiple items per receipt

ReceiptItem

- Product details (name, model, serial, IMEI)
- Price, quantity
- Product specifications (color, storage)
- Warranty information

Warranty (Receipt-based)

- Linked to ReceiptItem (1-to-1)
- Coverage period, provider, terms
- Auto-calculated expiry date

CustomerWarranty (Manual upload)

- Product name, expiry date
- Warranty image
- Independent of receipts

Claim

- Issue description, category
- AI-detected priority
- Status tracking
- Cost estimation

ClaimNote

- Internal notes for team communication

- Author tracking

ClaimAttachment

- File uploads (photos, documents)
- File size tracking

Notification

- Real-time updates
- Read/unread status
- Multiple notification types

Relationships

```
User (Customer) ─┬─→ Receipt ──→ ReceiptItem ──→ Warranty ──→ Claim
                 │                  │
                 └─→ CustomerWarranty              │
                                    │
User (Retailer) ─┬─→ Store ──→ Receipt              │
                 └──────────────────────────────────→ ClaimNote ←──────┘
                                    ClaimAttachment
```

⚡ Performance Optimization

Query Optimization Techniques

1. **Select Related** — Reduces N+1 queries

   ```
   Claim.objects.select_related(
       'warranty__receipt_item__receipt__customer'
   )
   ```

2. **Prefetch Related** — Efficient many-to-many

   ```
   Claim.objects.prefetch_related('notes', 'attachments')
   ```

3. **Subqueries** — Complex aggregations

```
latest_claim = Claim.objects.filter(
    warranty=OuterRef('pk')
).order_by('-submitted_at')

Warranty.objects.annotate(
    latest_claim_date=Subquery(
        latest_claim.values('submitted_at')[:1]
    )
)
```

4. **Conditional Aggregation** — Single query stats

```
Receipt.objects.aggregate(
    receipts_this_month=Count('id', filter=Q(date__gte=month_start)),
    receipts_last_month=Count('id', filter=Q(date__gte=last_month))
)
```

Performance Metrics

- **Before Optimization**: 66 queries (57 duplicates)
- **After Optimization**: 5–8 queries
- **Improvement**: ~90% reduction in database queries

Development Tools

```
# Query counter middleware
MIDDLEWARE = [
    'query_counter.middleware.QueryCounterMiddleware',
]

# Enable only in development
QUERY_COUNTER_ENABLED = DEBUG
```

🔒 Security

Authentication

- JWT tokens with refresh mechanism
- Access token expiry: 60 minutes
- Refresh token expiry: 1 day
- Secure password hashing (PBKDF2)

Authorization

- Role-based access control (RBAC)
- Object-level permissions
- Custom permission classes

Data Protection

- Environment variables for sensitive data
- `.gitignore` for secrets
- CORS configuration
- CSRF protection

File Upload Security

- File size limits (10MB max)
- File type validation
- Secure file storage
- Random file names

API Security

- Rate limiting (recommended for production)
- HTTPS in production
- SQL injection prevention (Django ORM)

- XSS protection

🤝 Contributing

We welcome contributions! Here's how:

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit changes (`git commit -m 'Add AmazingFeature'`)
4. Push to branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

Development Guidelines

- Follow PEP 8 style guide
- Write descriptive commit messages
- Add tests for new features
- Update documentation
- Optimize database queries

📄 License

This project is licensed under the MIT License.

📞 Support & Contact

- **GitHub Issues**: Report bugs or request features
- **Email**: kaxarovdev@gmail.com
- **Documentation**: This README and API docs at `/swagger/` and `/redoc/`

🎉 Acknowledgments

- **Hugging Face** — AI model hosting and inference
- **Django Community** — Excellent framework and documentation
- **Open Source Contributors** — All the amazing libraries we use

---

🚀 Roadmap

Planned Features

- [ ] Mobile app (React Native / Flutter)
- [ ] Email notifications
- [ ] SMS notifications
- [ ] Advanced OCR for receipt scanning
- [ ] Multi-language support
- [ ] Warranty marketplace
- [ ] Insurance integration
- [ ] Blockchain warranty verification
- [ ] QR code warranty cards
- [ ] Voice-based claim submission

Built with ❤️ by the xTeam

Last Updated: October 2025