Home Cleaning Service API

Complete Technical Documentation

Technology Stack:	FastAPI + MongoDB + Python
Architecture:	RESTful API with JWT Authentication
Project Type:	Marketplace Platform (Gig Economy)
Generated:	January 29, 2026

Table of Contents

1. Problem Statement

On-Demand Home Cleaning Service Marketplace

Overview

A digital marketplace platform that connects independent home cleaning professionals directly with customers seeking cleaning services. The platform empowers individual cleaners to operate as independent contractors while providing customers with transparent access to verified, rated cleaning services on-demand.

Problems Being Solved

For Home Cleaners:

- High commission fees charged by traditional cleaning agencies (30-40% of earnings)
- Lack of control over pricing and work schedules
- Limited direct customer relationships and repeat business opportunities
- No platform to showcase skills, ratings, and build professional reputation
- Difficulty reaching customers beyond local word-of-mouth referrals

For Customers:

- Difficulty finding trustworthy and reliable home cleaners
- Lack of transparency in pricing across different service providers
- No standardized quality metrics or verified customer reviews
- Unreliable scheduling and communication with cleaning services
- Limited options to compare services and choose based on specific needs

Proposed Solution

A gig-economy marketplace platform where individual home cleaners create professional profiles, set their own rates, define service packages, and manage their availability. Customers can search for cleaners based on location, ratings, pricing, and availability, then book services directly through the platform. The system facilitates trust through verified profiles, customer reviews, secure payments, and takes a minimal platform commission (10-15%) to sustain operations.

Key Platform Features

- **Dual User Roles:** Separate interfaces for customers and independent cleaner service providers
- **Dynamic Pricing Model:** Each cleaner sets their own hourly rates and service package prices

- **Real-time Availability:** Cleaners can toggle availability status and manage booking schedules
- **Location-Based Discovery:** Customers find cleaners within their geographic area using geospatial search
- **Integrated Payments:** Secure payment processing with automatic commission split
- **Trust & Transparency:** Rating and review system, verified profiles, booking history
- **Service Customization:** Cleaners create custom service packages (basic clean, deep clean, specific areas)
- **Communication:** In-app messaging between customers and cleaners for coordination

Success Metrics

- Enable cleaners to increase net earnings by 25-35% compared to traditional agency employment
- Provide customers with 3+ verified cleaner options within 5km radius
- Maintain platform average rating of 4.5+ stars across all services
- Achieve booking confirmation within 2 hours of customer request
- Support 100+ concurrent users with API response times under 500ms
- Reach 80% repeat customer rate within 6 months of platform launch

2. Project Folder Structure

The project follows a modular, layered architecture pattern that separates concerns into distinct packages. This structure promotes code reusability, maintainability, and follows industry best practices for FastAPI applications.

Directory/File	Purpose
app/	Main application package containing all source code
main.py	FastAPI application entry point, router registration, middleware setup
config.py	Centralized configuration management using Pydantic Settings
database.py	MongoDB connection setup using Motor async driver
app/models/	Database document structure definitions
user.py	User account document model (email, password, role)
cleaner.py	Cleaner profile document (bio, rates, availability, location)
booking.py	Booking document (dates, status, pricing snapshot)
service.py	Service package document (name, price, duration)
review.py	Review document (rating, comment, timestamps)
payment.py	Payment transaction document (amount, status, gateway info)
app/schemas/	Pydantic models for request/response validation
auth.py	Login, registration, token schemas
user.py	User profile update and response schemas
cleaner.py	Cleaner profile creation and update schemas
booking.py	Booking creation, update, and response schemas
service.py	Service package CRUD schemas
review.py	Review creation and listing schemas
common.py	Shared schemas (pagination, error responses)
app/routes/	API endpoint controllers (route handlers)
auth.py	Authentication endpoints (/api/auth/register, /login, /refresh)
users.py	User management endpoints (/api/users/me, profile updates)
cleaners.py	Cleaner discovery and profile endpoints
bookings.py	Booking creation, management, status updates
services.py	Service package CRUD operations
reviews.py	Review submission and retrieval
payments.py	Payment processing and verification

Directory/File	Purpose
app/services/	Business logic layer (separated from routes)
auth_service.py	Authentication logic (password verification, token generation)
user_service.py	User account operations
cleaner_service.py	Cleaner profile management and search logic
booking_service.py	Booking workflow orchestration and validation
payment_service.py	Payment gateway integration and processing
notification_service.py	Email/SMS notification delivery
app/utils/	Utility and helper functions
security.py	Password hashing (bcrypt), JWT encoding/decoding
dependencies.py	FastAPI dependency injection functions
validators.py	Custom validation logic
helpers.py	General-purpose helper functions
app/middleware/	Custom middleware components
error_handler.py	Global exception handling and error formatting
logging.py	Request/response logging middleware
rate_limiter.py	API rate limiting implementation
app/core/	Core application functionality
exceptions.py	Custom exception definitions
constants.py	Application-wide constants and enums
tests/	Test suite for all components
conftest.py	Pytest configuration and fixtures
test_auth.py	Authentication endpoint tests
test_cleaners.py	Cleaner functionality tests
test_bookings.py	Booking workflow tests
test_services.py	Service CRUD tests
Root Files	
.env	Environment variables (MongoDB URL, secrets) - NOT in version control
.env.example	Example environment file template for developers
.gitignore	Git ignore rules (excludes .env, __pycache__, etc.)

Directory/File	Purpose
requirements.txt	Python package dependencies
README.md	Project documentation and setup instructions
Dockerfile	Docker container configuration (optional)
docker-compose.yml	Multi-container Docker setup (optional)

Architectural Benefits of This Structure

- **Separation of Concerns:** Routes handle HTTP, services handle business logic, models define data structure
- **Testability:** Each layer can be tested independently with mocked dependencies
- **Scalability:** Easy to add new features by creating new modules without affecting existing code
- **Maintainability:** Clear organization makes it easy for new developers to understand and contribute
- **Reusability:** Business logic in services can be reused across multiple routes
- **Security:** Centralized authentication and validation through dependencies and middleware

3. System Architecture Overview

Three-Tier Architecture Pattern

The application follows a classic three-tier architecture pattern that separates presentation, business logic, and data management into distinct layers. This design promotes modularity, scalability, and maintainability.

Layer	Component	Responsibilities
Presentation	Frontend Application (React/Vue/Angular)	• User interface and interactions • Form validation • API request handling • State management
Business Logic	FastAPI Backend	• Request routing and validation • Authentication & authorization • Business rules enforcement • Data orchestration • External service integration
Data	MongoDB Database	• Persistent data storage • Document relationships • Data indexing • Query optimization

Complete Technology Stack

Category	Technology	Purpose & Rationale
Backend Framework	FastAPI	High-performance async framework with automatic API documentation
Database	MongoDB	Flexible NoSQL database for rapid development and schema evolution
DB Driver	Motor	Async MongoDB driver optimized for FastAPI async operations
Validation	Pydantic	Automatic data validation using Python type hints
Authentication	JWT (python-jose)	Stateless token-based authentication for scalability
Password Security	Bcrypt (passlib)	Industry-standard password hashing algorithm
Payment Processing	Razorpay/Stripe	Secure payment gateway integration for Indian/Global markets
Email/SMS	SendGrid/Twilio	Notification delivery services
Testing	Pytest + HTTPX	Comprehensive unit and integration testing
Server	Uvicorn	ASGI server for running async FastAPI application
Deployment	Docker	Containerization for consistent deployment across environments

RESTful API Design Principles

- **Resource-Based URLs:** /api/cleaners, /api/bookings (nouns, not verbs)

• **HTTP Methods:** GET (read), POST (create), PUT/PATCH (update), DELETE (remove)

• **Stateless Communication:** Each request contains all necessary information

• **Status Codes:** 200 (success), 201 (created), 400 (bad request), 401 (unauthorized), 404 (not found), 500 (server error)

• **JSON Format:** All requests and responses use JSON for data interchange

• **Versioning:** API versioning through URL prefix (/api/v1/)

• **Pagination:** Large datasets returned with page limits and offset parameters

4. Authentication & Authorization Flow

JWT (JSON Web Token) Authentication System

The platform uses JWT-based stateless authentication, where users receive signed tokens after successful login. These tokens are included in subsequent API requests to prove identity without server-side session storage.

1. User Registration Flow

1. User submits registration form with email, password, phone, and role (customer or cleaner)
2. Frontend sends POST request to /api/auth/register with form data
3. Backend validates data using Pydantic schemas (email format, password strength, phone format)
4. Password is hashed using bcrypt with cost factor 12 (approximately 250ms computation time)
5. New user document created in MongoDB 'users' collection with hashed password
6. Success response returned to frontend (user must login separately for security)
7. If cleaner role selected, prompt to complete profile setup after login

2. User Login Flow

1. User enters email and password on login page
2. Frontend sends POST request to /api/auth/login with credentials
3. Backend queries MongoDB users collection by email address
4. If user found, password verified using bcrypt.verify() against stored hash
5. On successful verification, backend generates two JWT tokens:
6. - **Access Token:** Short-lived (30 minutes), used for API requests, contains user_id and role
7. - **Refresh Token:** Long-lived (7 days), used only to obtain new access tokens
8. Both tokens returned to frontend in JSON response
9. Frontend stores tokens securely (httpOnly cookies or localStorage with XSS protection)
10. User redirected to appropriate dashboard (customer or cleaner view)

3. Authenticated API Request Flow

1. Frontend makes API request including Authorization header: 'Bearer <access_token>'
2. FastAPI dependency function intercepts request before route handler
3. Token extracted from Authorization header and validated for signature and expiration
4. Token payload decoded to extract user_id and role information
5. User details fetched from MongoDB to verify account still exists and is active

6. User object injected into route handler function via dependency injection
7. Route handler processes request with full user context
8. If token invalid/expired, return 401 Unauthorized; if user lacks permission, return 403 Forbidden

4. Token Refresh Flow

1. Access token expires after 30 minutes (detected when API returns 401)
2. Frontend automatically sends POST request to /api/auth/refresh with refresh token
3. Backend validates refresh token signature and expiration
4. If valid, new access token generated and returned
5. Frontend updates stored access token and retries failed request
6. User continues working without re-login interruption
7. If refresh token also expired, user must login again

5. Role-Based Authorization

The system implements role-based access control (RBAC) to restrict actions based on user type. Two primary roles exist: Customer and Cleaner, each with specific permissions.

Action	Customer	Cleaner
Create booking	✓ Yes	✗ No
Accept/reject booking	✗ No	✓ Own bookings only
Create service package	✗ No	✓ Yes
Update service pricing	✗ No	✓ Own services only
Leave review	✓ After booking completion	✗ No
Toggle availability	✗ No	✓ Yes
View all bookings	✓ Own bookings only	✓ Own bookings only
Update profile	✓ Own profile	✓ Own profile

5. Complete Booking Workflow

The booking process is the core transaction flow of the platform, involving multiple steps and state transitions. This section details the complete end-to-end workflow from cleaner discovery to service completion.

Step 1: Cleaner Discovery & Selection

Customer Action: Searches for cleaners in their area

API Request: GET /api/cleaners?city=Mumbai&available;=true&min;_rating=4.0&sort;=rating

Backend Processing:

- Queries MongoDB cleaners collection with filters
- Uses geospatial index for location-based search (within X km radius)
- Filters by availability status and rating threshold
- Aggregates average rating from reviews collection
- Sorts results by rating or distance

Response: List of matching cleaners with profiles, ratings, base rates, and services offered

Customer Action: Views detailed cleaner profile including:

- Biography and experience
- Available service packages with pricing
- Customer reviews and ratings
- Availability schedule
- Average response time

Step 2: Availability Verification

Customer Action: Selects service package and desired date

API Request: GET /api/cleaners/{cleaner_id}/availability?date=2026-02-10

Backend Processing:

- Retrieves cleaner's availability schedule for specified day
- Queries bookings collection for existing confirmed bookings on that date
- Calculates available time slots by subtracting booked periods from schedule
- Considers service duration when calculating slot availability

Response: List of available time slots (e.g., 9:00-12:00, 14:00-17:00)

Frontend: Displays available slots as interactive time picker

Step 3: Booking Creation

Customer Action: Fills booking form with:

- Selected service package
- Chosen date and time slot
- Service address (may differ from customer's profile address)
- Special instructions (e.g., 'Focus on kitchen', 'Pet in home')

API Request: POST /api/bookings

Request Body: {cleaner_id, service_id, booking_date, start_time, service_address, instructions}

Backend Validation:

- Verify customer is authenticated
- Confirm service belongs to selected cleaner
- Check cleaner still available at requested time (race condition protection)
- Validate address format and required fields
- Calculate total price: service_price + platform_fee (10%)

Database Operations:

- Create booking document with status='pending'
- Store pricing snapshot (in case cleaner changes rates later)
- Generate unique booking ID

Notifications:

- Send email/SMS to cleaner about new booking request
- Send confirmation email to customer

Response: Booking ID, total price, status, estimated confirmation time

Step 4: Cleaner Accepts or Rejects Booking

Cleaner Notification: Receives push notification, email, and sees request in dashboard

Cleaner Action: Reviews booking details and decides to accept or reject

API Request (Accept): PATCH /api/bookings/{booking_id}/status

Request Body: {status: 'confirmed'}

Backend Processing (Accept):

- Verify requester is the assigned cleaner
- Confirm booking still in 'pending' status
- Update status to 'confirmed'
- Create payment intent with payment gateway (Razorpay/Stripe)
- Send confirmation to customer with payment link

API Request (Reject): PATCH /api/bookings/{booking_id}/status

Request Body: {status: 'cancelled', reason: 'Schedule conflict'}

Backend Processing (Reject):

- Update status to 'cancelled'
- Record cancellation reason and timestamp
- Notify customer of cancellation
- Suggest alternative cleaners or time slots

Step 5: Payment Processing

Customer Action: Clicks payment link in confirmation email or pays through app

Payment Flow:

- Frontend initiates payment gateway SDK (Razorpay Checkout)
- Customer completes payment via UPI/card/netbanking
- Gateway processes payment and sends webhook to backend

Webhook Processing:

- Backend receives payment success notification
- Verifies payment signature to prevent fraud
- Updates payment_status to 'completed' in payments collection
- Updates booking payment_status
- Sends payment receipt to customer
- Notifies cleaner that payment received and booking confirmed

Commission Split:

- Platform fee (10%) held in platform account
- Cleaner earnings (90%) scheduled for payout after service completion

Step 6: Service Day Operations

Pre-Service Reminders:

- 24 hours before: Email reminder to both parties
- 1 hour before: SMS/push notification to cleaner
- 30 minutes before: Notification to customer

Day of Service:

- Cleaner marks status as 'in_progress' when starting
- Customer can see live status in app
- In-app chat available for coordination

Service Completion:

- Cleaner marks booking as 'completed' when finished
- System records completion timestamp

- Triggers automatic workflow for payment release and review request

Step 7: Review & Rating

Customer Notification: Prompted to leave review within 24 hours of completion

Customer Action: Submits rating (1-5 stars) and optional written review

API Request: POST /api/bookings/{booking_id}/review

Request Body: {rating: 5, comment: 'Excellent service, very thorough!'}

Backend Processing:

- Verify booking is completed and customer hasn't already reviewed
- Create review document in reviews collection
- Update cleaner's average rating using aggregation pipeline
- Increment cleaner's total review count
- Notify cleaner of new review

Payout Trigger:

- After review submitted or 48 hours pass (whichever comes first)
- Cleaner earnings released via payment gateway payout API
- Platform commission retained
- Payout confirmation sent to cleaner

Booking Status State Machine

Bookings transition through defined states with specific allowed transitions:

Status	Description	Next Possible States
pending	Awaiting cleaner response	confirmed, cancelled
confirmed	Cleaner accepted, payment pending/completed	in_progress, cancelled
in_progress	Service currently being performed	completed, cancelled
completed	Service finished successfully	None (terminal state)
cancelled	Booking cancelled by customer or cleaner	None (terminal state)

6. Database Schema & Collections

MongoDB is used for its flexible document model, allowing schema evolution as requirements change. The database consists of six primary collections with defined relationships and indexes.

1. Users Collection

Stores core account information for all platform users.

Field	Type	Description
_id	ObjectId	Unique identifier (auto-generated)
email	string	User email address (unique, indexed)
password_hash	string	Bcrypt hashed password (never plain text)
role	string	User type: "customer" or "cleaner" (indexed)
phone	string	Contact phone number
profile_pic	string	URL to profile image
is_active	boolean	Account status (for suspensions)
email_verified	boolean	Email verification status
created_at	datetime	Account creation timestamp
updated_at	datetime	Last profile update timestamp

Indexes: email (unique), role

2. Cleaner Profiles Collection

Extended profile information for users with 'cleaner' role.

Field	Type	Description
_id	ObjectId	Unique identifier
user_id	ObjectId	Reference to users collection (unique, indexed)
bio	string	Professional biography (max 500 chars)
experience_years	integer	Years of cleaning experience
hourly_rate	decimal	Base hourly rate in local currency
is_available	boolean	Current availability toggle (indexed)
availability_schedule	object	Weekly schedule with time slots per day
location.type	string	GeoJSON type: "Point"
location.coordinates	array	[longitude, latitude] for geospatial queries
address	object	{street, city, state, pincode}

verified	boolean	Background check completion status
rating_avg	decimal	Average rating (0-5, indexed)
rating_count	integer	Total number of ratings received
total_bookings	integer	Lifetime completed bookings count
created_at	datetime	Profile creation timestamp

Indexes: user_id (unique), is_available, location (2dsphere), rating_avg

3. Services Collection

Service packages created and priced by individual cleaners.

Field	Type	Description
_id	ObjectId	Unique identifier
cleaner_id	ObjectId	Reference to cleaner_profiles (indexed)
name	string	Service name (e.g., "Deep Kitchen Clean")
description	string	Detailed service description
price	decimal	Fixed price for this service package
duration_hours	decimal	Estimated duration (for scheduling)
service_type	string	Category: basic_clean, deep_clean, specialized
is_active	boolean	Whether cleaner still offers this service
created_at	datetime	Service creation timestamp

Indexes: cleaner_id, is_active

4. Bookings Collection

Core transaction records for all service bookings.

Field	Type	Description
_id	ObjectId	Unique identifier
customer_id	ObjectId	Reference to users (indexed)
cleaner_id	ObjectId	Reference to cleaner_profiles (indexed)
service_id	ObjectId	Reference to services
booking_date	datetime	Scheduled service date (indexed)
start_time	string	Start time (e.g., "10:00")
end_time	string	Calculated end time based on duration
service_price	decimal	Price snapshot at booking time
platform_fee	decimal	Commission amount (typically 10-15%)
total_price	decimal	service_price + platform_fee

status	string	pending, confirmed, in_progress, completed, cancelled (indexed)
service_address	object	Where service will be performed
special_instructions	string	Customer notes for cleaner
cancellation_reason	string	Reason if status=cancelled
cancelled_by	ObjectId	User who initiated cancellation
cancelled_at	datetime	Cancellation timestamp
payment_status	string	pending, completed, refunded
created_at	datetime	Booking request timestamp
updated_at	datetime	Last status change timestamp

Indexes: customer_id, cleaner_id, booking_date, status, compound (cleaner_id + booking_date + status)

5. Reviews Collection

Customer feedback and ratings for completed services.

Field	Type	Description
_id	ObjectId	Unique identifier
booking_id	ObjectId	Reference to bookings (unique - one review per booking)
customer_id	ObjectId	Reference to users
cleaner_id	ObjectId	Reference to cleaner_profiles (indexed)
rating	integer	Overall rating (1-5 stars)
comment	string	Written review (optional, max 1000 chars)
created_at	datetime	Review submission timestamp (indexed)

Indexes: booking_id (unique), cleaner_id + created_at (for recent reviews)

6. Payments Collection

Financial transaction records and payout tracking.

Field	Type	Description
_id	ObjectId	Unique identifier
booking_id	ObjectId	Reference to bookings (indexed)
customer_id	ObjectId	Reference to users
cleaner_id	ObjectId	Reference to cleaner_profiles
amount	decimal	Total payment amount
platform_fee	decimal	Platform commission
cleaner_earnings	decimal	Amount paid to cleaner after commission
payment_method	string	razorpay, stripe, cash, etc.

payment_status	string	pending, completed, failed, refunded (indexed)
transaction_id	string	Payment gateway transaction ID
payment_gateway_response	object	Full response from payment gateway
payout_status	string	pending, processed, failed
payout_date	datetime	When cleaner earnings were transferred
created_at	datetime	Payment initiation timestamp
updated_at	datetime	Last status change timestamp

Indexes: booking_id, payment_status, payout_status

7. API Endpoints Reference

Complete RESTful API endpoint specification following standard HTTP methods and conventions.

Authentication Endpoints (/api/auth)

Method	Endpoint	Description	Auth Required
POST	/register	Create new user account	No
POST	/login	Authenticate and receive JWT tokens	No
POST	/refresh	Get new access token using refresh token	No
POST	/logout	Invalidate refresh token	Yes
POST	/forgot-password	Request password reset email	No
POST	/reset-password	Complete password reset with token	No
POST	/verify-email	Verify email with verification token	No

User Management Endpoints (/api/users)

Method	Endpoint	Description	Auth Required
GET	/me	Get current user profile	Yes
PUT	/me	Update current user profile	Yes
GET	/{user_id}	Get public user information	No
DELETE	/me	Delete user account	Yes

Cleaner Discovery & Profile Endpoints (/api/cleaners)

Method	Endpoint	Description	Auth Required
GET	/	Search cleaners with filters (location, rating, price)	No
GET	/{cleaner_id}	Get detailed cleaner profile	No
POST	/profile	Create/update cleaner profile	Yes (Cleaner)
PATCH	/availability	Toggle availability status	Yes (Cleaner)
GET	/{cleaner_id}/services	Get services offered by cleaner	No
GET	/{cleaner_id}/reviews	Get reviews for cleaner	No
GET	/{cleaner_id}/availability	Get available time slots	No
GET	/me/bookings	Get my bookings (as cleaner)	Yes (Cleaner)
GET	/me/earnings	Get earnings dashboard	Yes (Cleaner)

Service Package Endpoints (/api/services)

Method	Endpoint	Description	Auth Required
POST	/	Create new service package	Yes (Cleaner)
GET	/{service_id}	Get service details	No
PUT	/{service_id}	Update service package	Yes (Owner)
DELETE	/{service_id}	Delete service package	Yes (Owner)

Booking Management Endpoints (/api/bookings)

Method	Endpoint	Description	Auth Required
POST	/	Create new booking request	Yes (Customer)
GET	/	Get my bookings (customer or cleaner)	Yes
GET	/{booking_id}	Get specific booking details	Yes (Participant)
PATCH	/{booking_id}/status	Update booking status	Yes (Participant)
DELETE	/{booking_id}	Cancel booking	Yes (Participant)

Review Endpoints (/api/reviews)

Method	Endpoint	Description	Auth Required
POST	/bookings/{booking_id}/review	Submit review for completed booking	Yes (Customer)
GET	/cleaners/{cleaner_id}	Get all reviews for cleaner	No
GET	/{review_id}	Get specific review	No

Payment Endpoints (/api/payments)

Method	Endpoint	Description	Auth Required
POST	/create-intent	Create payment intent for booking	Yes (Customer)
POST	/webhook	Payment gateway webhook callback	No (Signed)
GET	/{payment_id}	Get payment details	Yes (Participant)
POST	/refund/{payment_id}	Process refund	Yes (Admin)

8. Payment Processing Flow

The platform integrates with payment gateways (Razorpay for India, Stripe globally) to handle secure transactions and automatic commission splits between cleaners and the platform.

• **Booking Confirmation:** After cleaner accepts booking, payment intent created with gateway API. Platform receives payment_id and stores in payments collection with status='pending'.

• **Customer Payment:** Customer redirected to payment gateway page or SDK loads in app. Customer completes payment via UPI, card, netbanking, or wallet. Payment gateway processes transaction securely.

• **Webhook Notification:** Payment gateway sends webhook to backend endpoint /api/payments/webhook with payment status. Backend verifies webhook signature to prevent fraud. Payment document updated to status='completed'.

• **Booking Confirmation:** Booking payment_status updated to 'completed'. Confirmation emails/SMS sent to both customer and cleaner. Cleaner notified that booking is fully confirmed.

• **Escrow Period:** Platform holds funds in escrow until service completion. This protects both parties - customer can request refund if service not delivered, cleaner assured of payment.

• **Service Completion:** After booking marked 'completed' and review period passed (48 hours), payout workflow triggered automatically.

• **Commission Split:** Total amount split: Platform fee (10-15%) retained in platform account. Cleaner earnings (85-90%) transferred to cleaner's linked bank account.

• **Payout Processing:** Backend calls payment gateway payout API. Funds transferred to cleaner via bank transfer (typically 1-3 business days). Payout status updated to 'processed' with payout_date recorded.

• **Financial Reconciliation:** Both parties receive transaction receipts. Platform maintains audit trail of all transactions for tax and compliance purposes.

Refund & Cancellation Policy

• **Customer Cancellation (before service):** Full refund if cancelled 24+ hours before scheduled time. 50% refund if cancelled 6-24 hours before. No refund if cancelled less than 6 hours before.

• **Cleaner Cancellation:** Full refund issued immediately. Cleaner may receive penalty or rating impact. Customer offered alternative cleaner suggestions.

• **Service Issues:** If customer reports service quality issues, dispute resolution process initiated. Platform may issue partial or full refund based on investigation. Cleaner has opportunity to respond to complaints.

9. Security Best Practices

Security is paramount in a marketplace platform handling personal data and financial transactions. The following security measures are implemented throughout the application.

Password Security

- **Hashing Algorithm:** Bcrypt with cost factor 12 (approximately 250ms computation time, resistant to brute force)
- **Never Store Plain Text:** Passwords immediately hashed upon registration, never stored or logged in plain text
- **Password Requirements:** Minimum 8 characters, must include uppercase, lowercase, number, and special character
- **Password Reset:** Time-limited reset tokens (valid 1 hour), single-use only, invalidated after use
- **Account Lockout:** After 5 failed login attempts, account locked for 15 minutes or until email verification

JWT Token Security

- **Short Access Token Lifetime:** 30 minutes expiration reduces impact if token stolen
- **Secure Storage:** Tokens stored in httpOnly cookies (not accessible via JavaScript) or encrypted localStorage
- **Token Rotation:** New refresh token issued with each access token refresh, old refresh token invalidated
- **Signature Verification:** All tokens verified with secret key, tampering detected immediately
- **Logout:** Refresh token blacklisted in database on logout, cannot be reused

API Security

- **HTTPS Only:** All API requests encrypted with TLS 1.2+, no plain HTTP allowed
- **CORS Configuration:** Only whitelisted frontend origins allowed to make requests
- **Rate Limiting:** Maximum 100 requests per minute per IP address, prevents abuse and DDoS
- **Input Validation:** All inputs validated with Pydantic before processing, prevents injection attacks
- **SQL/NoSQL Injection Prevention:** Parameterized queries and schema validation prevent malicious inputs
- **XSS Protection:** All user-generated content sanitized before storage and display

Data Privacy & Protection

- **Sensitive Data Exclusion:** API responses never include password hashes, payment tokens, or internal IDs
- **Role-Based Access:** Users can only access their own data and public information of others

- **Payment Data:** Credit card details never stored on our servers, handled entirely by payment gateway
- **Personal Information:** Phone numbers, addresses shown only to participants in confirmed bookings
- **Data Encryption:** Sensitive fields encrypted at rest in database
- **GDPR Compliance:** Users can request data export or account deletion, honored within 30 days

Payment Security

- **PCI-DSS Compliance:** Payment processing handled by certified gateway, no card data on our infrastructure
- **Webhook Verification:** All payment webhooks verified via signature, prevents spoofed callbacks
- **Fraud Detection:** Unusual transaction patterns flagged for manual review
- **Refund Authorization:** Refunds require admin approval and are logged in audit trail
- **Transaction Reconciliation:** Daily automated reconciliation between database records and gateway statements

Security Monitoring & Incident Response

- **Logging:** All authentication attempts, failed logins, and permission denials logged
- **Alert System:** Automated alerts for suspicious patterns (multiple failed logins, unusual API access)
- **Regular Audits:** Weekly security audits of logs and access patterns
- **Vulnerability Scanning:** Automated dependency scanning for known security vulnerabilities
- **Incident Response Plan:** Documented procedures for data breach response, including user notification

--- End of Documentation ---

This documentation provides a comprehensive technical overview of the Home Cleaning Service API platform. For implementation details and code examples, refer to the project repository README and inline code documentation.