Community Tourist Assistant Technical Documentation

Release 1.0

Community Tourist Assistant Team

Feb 10, 2026

Welcome to the technical documentation for the Community Tourist Assistant project.

This documentation explains how the platform supports a crowd-sourced tourism model with low staff overhead, high content quality controls, and user-friendly journeys for guests, contributors, and moderators.

Chapter 1

Project Context

Many communities depend on tourism but face staffing and budget constraints. This platform uses community-submitted place content, reviews, and moderation workflows to increase local discovery while keeping operational overhead manageable.

1.1 Architecture

1.1.1 Overview

The application is split into focused Django apps:

- `accounts`: authentication, profile, contribution scoring
- `places`: place models, submission/search/detail workflows
- `reviews`: review submission/reporting/moderation
- `community_tourism`: project-level routing and home views

1.1.2 Layered Structure

The codebase follows a practical layered approach:

- **Presentation layer**: Django templates and view functions
- **Application layer**: view orchestration, forms, moderation rules
- **Domain layer**: models encapsulating key behavior (e.g., opening hours, trust logic)
- **Data layer**: PostgreSQL persistence, indexes, migration-based schema evolution

This separation keeps each area understandable and supports iterative MVP growth.

1.1.3 Data Model Highlights

- `Place` uses polymorphic inheritance for type-specific fields.
- `PlaceLike` is an explicit model for flexible like metadata/analytics.
- `ReviewReport` stores user reports with moderation status.
- `ModerationLog` stores an audit trail for admin moderation actions.

1.1.4 Key Relationships

- A `Place` has many `Review` and `PlaceImage` entries.
- A `Place` has many `PlaceLike` entries (explicit join model).

- A `Review` can have many `ReviewReport` entries.
- `ModerationLog` references moderated objects generically for a unified audit trail.

1.1.5 Why this Architecture Fits the Scenario

- Supports public browsing and quick read performance (indexed queries).
- Supports trusted crowd contribution with moderation checkpoints.
- Supports low-overhead operations via admin actions and bulk moderation.
- Supports future growth (new place subtypes, analytics, API layer) without redesign.

1.2 Design Patterns and Justifications

1.2.1 Polymorphic Model Pattern

`Place` is a polymorphic base model with subtype models such as `HeritagePlace`, `FoodPlace`, `ActivityPlace`, and `BeachPlace`.

Why used:

- Shared fields stay centralized.
- Type-specific fields stay clean and explicit.
- UI can render subtype details without large conditional model structures.

1.2.2 Repository via ORM + Query Composition

Django ORM querysets are composed in views for search, moderation queues, and detail pages.

Why used:

- Improves readability of filtering and sorting logic.
- Allows incremental optimization via annotations and indexes.
- Keeps SQL portability while still supporting query plan verification.

1.2.3 Audit Trail Pattern

`ModerationLog` records moderation actions with actor, action type, target object, and timestamp.

Why used:

- Provides accountability for admin actions.
- Supports governance/reporting requirements.
- Helps explain content state changes during stakeholder demos.

1.2.4 Soft Delete Pattern

Places and reviews use archive fields instead of hard deletes.

Why used:

- Preserves moderation evidence.
- Enables reversible operations (restore).
- Reduces risk of accidental data loss.

1.2.5 Progressive Trust Pattern

Contribution points and review restrictions provide graduated control.

Why used:

- Encourages healthy participation.
- Adds non-binary moderation responses.
- Scales better than manual intervention alone.

1.3 Features

1.3.1 Feature Objectives

The feature set is designed to satisfy five scenario outcomes:

1. Public browsing and discovery
2. Community ratings and reviews
3. Registered user contributions
4. Encouraging participation through progression
5. Low-overhead moderation for administrators

1.3.2 Core User Features

- Browse approved places and view details
- Advanced search with filtering and sorting
- Add places with type-specific attributes and image uploads
- Add reviews and report inappropriate content
- Like/unlike places

1.3.3 Content Quality Features

- Place submission moderation statuses
- Review reporting queue and admin moderation actions
- Contribution points and trust progression
- Anti-spam protections (rate limiting, honeypot, CAPTCHA escalation, similarity checks)

1.3.4 Data and Discovery Quality

- Rich place metadata (location, contact, visitor essentials)
- Opening-hours model with open-now computation (where applicable)
- Thumbnail-first result cards for quick scanning
- Query annotations for like/review/rating context

1.4 Usability by User Type

1.4.1 Guest Visitors

Primary goals:

- Discover interesting local places quickly.
- Compare quality and relevance.
- Decide where to visit next.

Current support:

- Home page with top-rated and socially validated places.
- Search filters (category, rating, images, open-now, sorting).
- Rich place detail pages (photos, opening status, visitor essentials).

1.4.2 Registered Contributors

Primary goals:

- Add useful place entries and media.
- Share reviews.
- Build trust/recognition over time.

Current support:

- Guided add-place flow with optional advanced metadata.
- Image previews before upload.
- Contribution scoring and trust levels.
- Clear submission states (pending/approved/rejected).

1.4.3 Moderators / Admins

Primary goals:

- Keep content quality high with low operational effort.
- Resolve reports quickly.
- Maintain transparency and consistency.

Current support:

- Bulk moderation actions for places and reviews.
- Dedicated review report handling paths.
- Soft archive/restore instead of destructive delete.
- Moderation audit logs for traceability.

1.4.4 Accessibility and UX Principles

- Keyboard-focus styling and skip links.
- Semantic headings and form labels.

- Clear status messaging for moderation and validation outcomes.
- Mobile-responsive layout with Bootstrap components.

1.5 Moderation Workflow

1.5.1 Place Moderation

1. Users submit places in pending state.
2. Admin approves/rejects/archives submissions.
3. Actions are recorded in `ModerationLog`.

1.5.2 Review Moderation

1. Users report reviews.
2. Admin upholds or dismisses reports.
3. Upheld reports can penalize contribution score and restrict posting.
4. Actions are recorded in `ModerationLog`.

1.5.3 Soft Delete Strategy

- `Place` and `Review` support archive fields.
- Admin delete paths archive items instead of hard delete.
- Archived content is excluded from public-facing queries.

1.6 Operations

1.6.1 Environment

Runtime configuration is loaded from environment variables in `.env`.

1.6.2 Database

Supported backends:

- SQLite (default/dev)
- PostgreSQL (recommended for production)

1.6.3 Deployment Notes

- Keep secrets in environment variables.
- Ensure media backup strategy is in place.
- Run migrations on each deployment.

1.6.4 Build Documentation

From repository root:

- HTML: `sphinx-build -b html docs/sphinx/source docs/sphinx/_build/html`
- PDF (rinoh): `sphinx-build -b rinoh docs/sphinx/source docs/sphinx/_build/rinoh`

1.7 Deployment Guide

This guide walks through deploying the Community Tourist Assistant from scratch. It covers local setup, PostgreSQL configuration, and a production deployment using Gunicorn and Nginx. Adjust hostnames and paths to match your environment.

1.7.1 Prerequisites

- Python 3.11+ and pip
- PostgreSQL 15+ (or SQLite for local-only testing)
- Git
- A Linux server (for production) with systemd

1.7.2 Local Setup (Development)

1. Clone the repository and create a virtual environment.

```
git clone <repo-url>
cd SoftwareEngineering1
python -m venv venv
.\venv\Scripts\activate
```

2. Install dependencies.

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

3. Create a local `.env` file (copy `.env.example`).

```
copy .env.example .env
```

4. Apply migrations and create an admin account.

```
python manage.py migrate
python manage.py createsuperuser
```

5. Run the development server.

```
python manage.py runserver
```

1.7.3 PostgreSQL Setup

1. Create database and user.

```
psql -U postgres
CREATE DATABASE community_tourism;
CREATE USER tourism_user WITH PASSWORD 'your_password';
GRANT ALL PRIVILEGES ON DATABASE community_tourism TO tourism_user;
```

2. Update `.env` with PostgreSQL credentials.

```
DB_ENGINE=django.db.backends.postgresql
DB_NAME=community_tourism
DB_USER=tourism_user
DB_PASSWORD=your_password
DB_HOST=localhost
DB_PORT=5432
```

3. Apply migrations.

```
python manage.py migrate
```

1.7.4 Static and Media Files

1. Set static and media paths in `settings.py` (already configured).
2. Collect static assets for production.

```
python manage.py collectstatic
```

1.7.5 Production Deployment (Gunicorn + Nginx)

1. Install system packages.

```
sudo apt update
sudo apt install python3-venv python3-pip nginx
```

2. Create application directory and virtual environment.

```
sudo mkdir -p /srv/community_tourism
sudo chown $USER:$USER /srv/community_tourism
cd /srv/community_tourism
git clone <repo-url> .
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

3. Configure environment variables.

```
cp .env.example .env
nano .env
```

4. Run migrations and collect static files.

```
python manage.py migrate
python manage.py collectstatic
```

5. Test Gunicorn.

```
gunicorn community_tourism.wsgi:application --bind 0.0.0.0:8000
```

1.7.6 Systemd Service

Create a systemd unit file at `/etc/systemd/system/community_tourism.service`.

```
[Unit]
Description=Community Tourist Assistant
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/srv/community_tourism
EnvironmentFile=/srv/community_tourism/.env
ExecStart=/srv/community_tourism/venv/bin/gunicorn
 community_tourism.wsgi:application --bind 127.0.0.1:8000

[Install]
WantedBy=multi-user.target
```

Enable and start the service.

```
sudo systemctl daemon-reload
sudo systemctl enable community_tourism
sudo systemctl start community_tourism
```

1.7.7 Nginx Configuration

Create an Nginx config at `/etc/nginx/sites-available/community_tourism`.

```
server {
    listen 80;
    server_name your-domain.com;

    location /static/ {
        alias /srv/community_tourism/staticfiles/;
    }

    location /media/ {
        alias /srv/community_tourism/media/;
    }

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Enable the site and reload Nginx.

```
sudo ln -s /etc/nginx/sites-available/community_tourism /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

1.7.8 HTTPS (Recommended)

Use Certbot to add a TLS certificate.

```
sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d your-domain.com
```

1.7.9 Operational Tasks

- Backups: schedule nightly PostgreSQL dumps.

- Logs: monitor Gunicorn and Nginx logs for errors.
- Updates: pull latest code, run migrations, restart the service.

1.8 Testing

1.8.1 Test Strategy

- Unit tests for model methods, validators, and utilities
- Integration tests for user and moderation flows
- Admin action tests for moderation controls

1.8.2 Commands

- Run all tests: `pytest -q`
- Coverage: `pytest --cov=accounts --cov=community_tourism --cov=places --cov=reviews --cov-report=term-missing -q`

1.8.3 Current Status

The project includes a high-coverage automated test suite with CI-friendly commands.

1.9 API Reference

This section is generated from docstrings across the application codebase.

1.10 API Reference (Generated)

This section is generated directly from docstrings.

1.10.1 Accounts

accounts.models

Models for tracking user contributions, trust levels, and moderation impacts.

Contribution

Track contribution counts, points, and trust levels for a user.

param user The user this contribution record belongs to.

param places_added
Count of approved place submissions by the user.

param reviews_added
Count of reviews posted by the user.

param points Total contribution points accumulated by the user.

param upheld_reports_count
Count of upheld reports against the user's reviews.

param review_restriction_active
Flag indicating if the user is blocked from posting reviews.

Contribution.is_trusted

Return True if the user's points meet the trusted threshold.

return True when points are at or above TRUSTED_THRESHOLD.

rtype bool

Contribution.level_name

Return the display name for the user's current contribution level.

return The current level name based on points.

rtype str

Contribution.level_badge_class

Return the Bootstrap badge class for the user's current level.

return The badge class string.

rtype str

Contribution.next_level_name

Return the next level name the user is working toward.

return The next level name or None if already at top level.

rtype str | None

Contribution.points_to_next_level

Return the number of points required to reach the next level.

return Points remaining to next level, or 0 if at top level.

rtype int

Contribution.level_progress_percent

Return the user's progress toward the next level as a percentage.

return Progress percentage from 0 to 100.

rtype int

Contribution.__str__

Return a human-readable summary of the contribution record.

return Summary string with username and points.

rtype str

accounts.views

Views for account authentication, activation, profile, and account management.

accounts.views.activate_account

Activate a newly created account using a time-limited token link.

param request Incoming HTTP request.

param uidb64 Base64 encoded user ID.

param token Activation token issued at signup.

return Redirect to login with a success or error message.

accounts.views.contributions_view

Render detailed contribution stats page.

param request Incoming HTTP request for an authenticated user.

return Rendered contributions summary page.

accounts.views.delete_account

Allow users to permanently delete their own account.

param request	Incoming HTTP request, must be POST to delete.
return	Rendered confirmation page or redirect after deletion.

accounts.views.login_view

Sign in an existing user account.

param request	Incoming HTTP request containing optional POST credentials.
return	Rendered login page or redirect on successful authentication.

accounts.views.logout_view

Log out the current user.

param request	Incoming HTTP request.
return	Redirect to home page after logout.

accounts.views.profile_view

Show profile details, contribution stats, and moderation outcomes.

param request	Incoming HTTP request for an authenticated user.
return	Rendered profile page with contribution and submission data.

accounts.views.signup_view

Create an inactive account and email an activation link.

param request	Incoming HTTP request with signup form data.
return	Rendered signup form on validation errors or redirect to login on success.

accounts.signals

Signal handlers for contribution tracking.

accounts.signals.award_points_for_place

Award contribution points when a user submits a place.

param sender	The Place model class.
param instance	The Place instance that was saved.
param created	True if the place was created in this save.
param kwargs	Additional signal keyword arguments.
return	None

accounts.signals.award_points_for_review

Award contribution points when a user posts a review.

param sender	The Review model class.
param instance	The Review instance that was saved.
param created	True if the review was created in this save.
param kwargs	Additional signal keyword arguments.
return	None

accounts.signals.create_contribution

Create a Contribution record when a new user is created.

param sender	The model class sending the signal.
param instance	The newly created User instance.
param created	True if the user was created in this save.
param kwargs	Additional signal keyword arguments.
return	None

1.10.2 Places

places.models

Models for places, place types, images, and likes.
ActivityPlace

Place subtype for activities with age and booking details.
BeachPlace

Place subtype for beaches and lakes with safety and facility info.
FoodPlace

Place subtype for food and drink venues with cuisine and dietary options.
HeritagePlace

Place subtype for heritage sites with historical details.
Place

Base model for all place types with shared attributes and moderation fields.
Place.likes_count

Return the total number of likes for this place.

return Count of PlaceLike records.

rtype int
Place.average_rating

Return the rounded average rating for this place.

return Average rating rounded to two decimals, or None if no reviews.

rtype float | None
Place.has_opening_hours

Return True when both opening and closing times are set.

return True if opening hours are present.

rtype bool
Place.supports_opening_hours

Return True if opening hours are applicable for this place type.

return False for beaches, True otherwise.

rtype bool
Place.opening_days_display

Return a human-readable display of opening days.

return Display string such as 'Mon, Wed, Fri' or 'Daily'.

rtype str
Place.is_open_now

Return whether the place is open at the current time.

return True if open now, False if closed, None if not applicable.

rtype bool | None
Place.__str__

Return the place name for display.

return Place name.

rtype str

Place.save

Save the place and synchronize is_approved from moderation_status.

param args Positional arguments forwarded to Model.save.

param kwargs Keyword arguments forwarded to Model.save.

return None

PlaceImage

Image associated with a place.

PlaceImage.__str__

Return a readable label for the image.

return Label with place name.

rtype str

PlaceLike

Like relationship between a user and a place.

PlaceLike.__str__

Return a readable label for the like.

return Label with user and place.

rtype str

places.models.validate_image_size

Validate image size for place uploads.

param image Uploaded image file.

return None if valid.

raises ValidationError

If the image exceeds the size limit.

places.forms

Forms for creating and editing places and their opening hours.

ActivityPlaceForm

Form for activity place-specific fields.

ActivityPlaceForm.media

No documentation available.

BeachPlaceForm

Form for beach place-specific fields.

BeachPlaceForm.media

No documentation available.

FoodPlaceForm

Form for food place-specific fields.

FoodPlaceForm.media

No documentation available.

HeritagePlaceForm

Form for heritage place-specific fields.

HeritagePlaceForm.media

No documentation available.

MultipleFileInput

File input widget allowing multiple file selection.

MultipleFileInput.media

No documentation available.

PlaceForm

Base form for creating a Place with shared fields and opening hours.

PlaceForm.clean

Validate place submission, including location and opening hours rules.

return Cleaned form data with normalized opening hour fields.

rtype dict

raises forms.ValidationError

If location data or hours are incomplete.

PlaceForm.media

No documentation available.

PlaceImageUploadForm

Form for uploading multiple place images.

PlaceImageUploadForm.media

No documentation available.

PlaceOpeningHoursForm

Form for editing only the opening hours fields on a Place.

PlaceOpeningHoursForm.__init__

Initialize the form with existing opening hour values.

param args Positional arguments passed to ModelForm.

param kwargs Keyword arguments passed to ModelForm.

return None

PlaceOpeningHoursForm.clean

Validate opening hours and normalize fields.

return Cleaned form data with opening hour fields normalized.

rtype dict

raises forms.ValidationError

If one time is missing.

PlaceOpeningHoursForm.save

Persist opening hours to the associated Place.

param commit Whether to save changes to the database.

return Updated Place instance.

rtype Place

PlaceOpeningHoursForm.media

No documentation available.

places.views

Views for browsing, searching, reviewing, and submitting places.

AddPlaceView

Create a new polymorphic place submission with optional images.

AddPlaceView.get

Render an empty add-place form for authenticated users.

param request Incoming HTTP request.

return Rendered add place page.

AddPlaceView.post

Handle add-place form submission and persist the new place.

param request Incoming HTTP request with form data and files.

return Redirect to place list on success or re-render form on error.

AddPlaceView.dispatch

No documentation available.

EditOpeningHoursView

Allow logged-in users to add opening hours when missing; owners/staff can update.

EditOpeningHoursView.get

Render the opening hours edit form when user has permission.

param request Incoming HTTP request.

param pk Place primary key.

return Rendered opening hours edit page or redirect on error.

EditOpeningHoursView.post

Handle opening hours updates and persist changes.

param request Incoming HTTP request with opening hour data.

param pk Place primary key.

return Redirect to place detail on success or re-render form on error.

PlaceDetailView

Show place details, reviews, metrics, and handle in-page review submissions.

PlaceDetailView.get

Render a place detail page with reviews, ratings, and nearby places.

param request Incoming HTTP request.

param pk Place primary key.

return Rendered place detail page.

PlaceDetailView.post

Handle review submissions from the place detail page.

param request Incoming HTTP request with review form data.

param pk Place primary key.

return Redirect back to place detail page.

PlaceDetailView.dispatch

No documentation available.

PlaceListView

List approved places for public browsing.

PlaceListView.get

Render the list of approved, non-archived places.

param request Incoming HTTP request.

return Rendered place list page.

SearchPlacesView

Search approved places with richer filtering, sorting, and pagination.

SearchPlacesView.get

Render the search page with filtered and sorted results.

param request Incoming HTTP request with filter query params.

return Rendered search results page.

ToggleLikeView

Toggle like state for a place. Returns JSON for AJAX requests.

ToggleLikeView.post

Toggle the current user's like for a place.

param request Incoming HTTP request.

param pk Place primary key.

return JSON response for AJAX or redirect for standard requests.

ToggleLikeView.get

Disallow GET on like toggles.

param request Incoming HTTP request.

param pk Place primary key.

return HTTP 405 response.

places.utils

Utility helpers for place-related services.

places.utils.geocode_location

Convert postcode or address into latitude and longitude using postcodes.io.

param location_text

Postcode or address string.

return Tuple of (latitude, longitude) or (None, None) if lookup fails.

rtype tuple[float | None, float | None]

1.10.3 Reviews

reviews.models

Models for reviews, reports, ratings, and moderation logs.

ModerationLog

Audit log entry for moderation actions on reviews or places.

ModerationLog.__str__

Return a readable label for the moderation log entry.

return Summary string describing the action and target.

rtype str

Rating

Legacy rating model for direct place scores.

Rating.__str__

Return str(self).

Review

User-submitted review of a place with moderation metadata.

Review.__str__

Return a readable label for the review.

return Summary string with user or guest label.

rtype str

ReviewReport

User report filed against a review for moderation.

ReviewReport.__str__

Return a readable label for the report.

return Summary string with review id and reporter.

rtype str

reviews.forms

Forms for creating user reviews with lightweight anti-spam fields.

ReviewForm

Review form with rating, text, honeypot, and optional CAPTCHA.

ReviewForm.__init__

Initialize the form and configure CAPTCHA when required.

param args Positional arguments passed to ModelForm.

param require_captcha Whether the security question should be required.

param captcha_question The question to display when CAPTCHA is required.

param kwargs Keyword arguments passed to ModelForm.

return None

ReviewForm.clean_honeypot

Validate the hidden honeypot field to detect bots.

return Empty string when no spam detected.

rtype str

raises forms.ValidationError If the honeypot field is filled.

ReviewForm.media

No documentation available.

reviews.views

Views for listing reviews, creating reviews, and reporting abusive reviews.

reviews.views.add_review

Create a review for a place with duplicate and restriction safeguards.

param request Incoming HTTP request with optional POST form data.

param place_id Place primary key.

return Rendered review form or redirect back to place detail.

reviews.views.place_reviews

Render approved reviews for a specific place.

param request Incoming HTTP request.

param place_id Place primary key.

return Rendered reviews list for the place.

reviews.views.report_review

Log a user report for a review and flag it for moderation.

param request Incoming HTTP request with report reason.

param review_id Review primary key.

return Redirect back to place detail with feedback.

reviews.spam

Anti-spam helpers for review submissions.

reviews.spam.get_or_create_captcha

Return the current CAPTCHA requirement and question for the session.

param request Incoming HTTP request.

return Tuple of (required flag, question text).

rtype tuple[bool, str]

reviews.spam.is_duplicate_or_similar_review

Check whether a review is a duplicate or too similar for a given place.

param place Place instance to compare against.

param text Review text to evaluate.

return True if duplicate or highly similar review is detected.

rtype bool

reviews.spam.normalize_text

Normalize text for similarity comparison.

param text Raw review text input.

return Normalized text with punctuation removed and whitespace collapsed.

rtype str

reviews.spam.require_captcha

Flag the session so CAPTCHA is required on the next submission.

param request Incoming HTTP request.

return None

reviews.spam.validate_captcha

Validate a submitted CAPTCHA answer and clear the requirement on success.

param request Incoming HTTP request.

param submitted_answer User's answer to the CAPTCHA question.

return True if the answer matches the expected value.

rtype bool

reviews.moderation

Shared moderation helpers for logging admin actions.

reviews.moderation.log_moderation_action

Create a moderation log entry for the given action and target.

param actor User performing the action.

param action Action enum value from ModerationLog.Action.

param target Model instance being moderated.

param notes Optional free-text notes for audit context.

return None

Python Module Index