Supermatcher

Release 1.0.0

Daniel Barreiros, David Pereira

Nov 12, 2025

CONTENTS:

Welcome to Supermatcher documentation!

CHAPTER

ONE

RUN_WEBSERVER MODULE

Run Webserver - Startup Script

This script provides an interactive first-time setup for the Biometric Webserver and launches the server. It allows you to create initial admin and endpoint accounts, configure SSL, and start the webserver with the desired settings.

`run_webserver.first_time_setup()`

Launches an interactive setup wizard for the first run of the Biometric Webserver.

Prompts the user to create admin and endpoint accounts, set passwords, and choose SSL options. Initializes the database and creates the required users.

Returns

(use_ssl, port) where use_ssl is a boolean indicating if HTTPS should be used,
and port is the selected port number.

Return type
tuple

`run_webserver.generate_self_signed_cert()`

Generates a self-signed SSL certificate and private key for HTTPS connections.

The certificate and key are saved to the default SSL certificate directory as defined in the configuration.

Returns

(certfile, keyfile) where certfile is the path to the generated certificate file,
and keyfile is the path to the generated private key file.

Return type
tuple

`run_webserver.main()`

Main entry point for starting the Biometric Webserver.

Handles command-line arguments for server configuration, checks if first-time setup is needed, allows resetting the admin password, configures SSL certificates, and launches the server using uvicorn.

CHAPTER

TWO

SRC

2.1 config module

Configuration file for Supermatcher v1.0 (Hybrid)

This module contains all configurable parameters for the fingerprint matching system.

Modify these values to tune the system behavior without changing the core code.

class config.**FusionConfig**(*enabled=True*, *distance=12.0*, *angle_deg=15.0*, *min_consensus=0.5*, *keep_raw=False*, *mode='optimal'*)

Bases: `object`

Configuration for template fusion.

angle_deg: float = 15.0

distance: float = 12.0

enabled: bool = True

keep_raw: bool = False

min_consensus: float = 0.5

mode: str = 'optimal'

config.**validate_config**()

Validate configuration consistency.

2.2 extractor module

Feature Extraction Module for Supermatcher v1.0 (Hybrid)

This module contains all feature extraction functions for fingerprint processing:

- Ridge orientation and frequency estimation
- Log-Gabor filtering for ridge enhancement
- Binarization and skeletonization
- Minutiae extraction and validation
- Optional Level-3 features (pores)

`extractor.`**`apply_log_gabor_enhancement`**(*image*, *mask*, *orientation*, *frequency*, *block_size=None*, *scales=(0.85, 1.0, 1.2)*)

Enhance fingerprint ridges using Log-Gabor filters.

Parameters

- **image** (*np.ndarray*) – Diffusion-enhanced image (float32, 0-255).
- **mask** (*np.ndarray*) – Boolean foreground mask.
- **orientation** (*np.ndarray*) – Ridge orientation map (radians).
- **frequency** (*np.ndarray*) – Ridge frequency map (cycles/pixel).
- **block_size** (*Optional[int], optional*) – Processing block size. Uses config default if None.
- **scales** (*Tuple[float, ...], optional*) – Frequency scale factors. Defaults to (0.85, 1.0, 1.2).

Returns

Enhanced fingerprint image (float32, 0-255).

Return type

np.ndarray

`extractor.`**`binarise_and_thin`**(*image*, *mask*)

Binarize enhanced image and compute skeleton.

Parameters

- **image** (*np.ndarray*) – Enhanced fingerprint image (float32, 0-255).
- **mask** (*np.ndarray*) – Boolean foreground mask.

Returns

Tuple of (binary, skeleton):

- binary: Binary image with ridges=1, valleys=0 (uint8).
- skeleton: Thinned 1-pixel wide skeleton (uint8).

Return type

Tuple[np.ndarray, np.ndarray]

`extractor.`**`create_log_gabor_kernel`**(*size*, *f0*, *theta0*, *sigma_r=1.5*, *sigma_theta_deg=12.0*)

Construct a Log-Gabor filter kernel in frequency domain.

Parameters

- **size** (*int*) – Kernel size (should match block_size).
- **f0** (*float*) – Center frequency in cycles/pixel.
- **theta0** (*float*) – Ridge orientation in radians.
- **sigma_r** (*float, optional*) – Radial bandwidth parameter. Defaults to 1.5.
- **sigma_theta_deg** (*float, optional*) – Angular bandwidth in degrees. Defaults to 12.0.

Returns

Complex-valued frequency-domain filter kernel.

Return type

np.ndarray

`extractor.`**`detect_pores`**`(`*image*, *mask*, *min_radius=None*, *max_radius=None*, *threshold=None*`)`

Detect sweat pores using multi-scale Laplacian-of-Gaussian (LoG) filter.

Parameters

- **`image`** (*`np.ndarray`*) – Enhanced fingerprint image (float32, 0-255).
- **`mask`** (*`np.ndarray`*) – Boolean foreground mask.
- **`min_radius`** (*`Optional[float], optional`*) – Minimum pore radius. Uses config default if None.
- **`max_radius`** (*`Optional[float], optional`*) – Maximum pore radius. Uses config default if None.
- **`threshold`** (*`Optional[float], optional`*) – Minimum detection strength. Uses config default if None.

Returns

List of Pore objects with (x, y, radius, strength).

Return type

List[*Pore*]

`extractor.`**`estimate_orientation_and_frequency`**`(`*image*, *mask*, *block_size=None*`)`

Estimate ridge orientation and spatial frequency for each block.

Parameters

- **`image`** (*`np.ndarray`*) – Enhanced fingerprint image (float32, 0-255).
- **`mask`** (*`np.ndarray`*) – Boolean foreground mask.
- **`block_size`** (*`Optional[int], optional`*) – Analysis block size. Uses config default if None.

Returns

Tuple of (orientation, frequency):

- orientation: Ridge angle in radians, shape (blocks_y, blocks_x).
- frequency: Spatial frequency in cycles/pixel, shape (blocks_y, blocks_x).

Return type

Tuple[np.ndarray, np.ndarray]

`extractor.`**`estimate_ridge_frequency`**`(`*block*, *theta*`)`

Estimate local ridge frequency from 1D projection orthogonal to orientation.

Parameters

- **`block`** (*`np.ndarray`*) – Image block (float32, typically 16Œ16 pixels).
- **`theta`** (*`float`*) – Ridge orientation in radians.

Returns

Ridge frequency in cycles per pixel (0.0 if invalid).

Return type

float

`extractor.`**`extract_minutiae`**`(`*skeleton*, *mask*, *orientation*, *block_size=None*`)`

Extract minutiae points using Crossing Number (CN) method.

Parameters

- **skeleton** (*np.ndarray*) – Thinned binary skeleton (uint8, 0 or 1).
- **mask** (*np.ndarray*) – Boolean foreground mask.
- **orientation** (*np.ndarray*) – Ridge orientation map (radians).
- **block_size** (*Optional[int], optional*) – Block size for orientation lookup. Uses config default if None.

Returns

List of Minutia objects with (x, y, angle, type, quality).

Return type

List[*Minutia*]

extractor.**validate_minutiae**(*minutiae*, *skeleton*, *mask*, *validate_with_context=True*)

Remove spurious minutiae using spatial and structural heuristics.

Filters out false minutiae caused by noise, broken ridges, or artifacts. Applies multiple validation criteria:

1. Mask-based: Remove minutiae in background regions.
2. Distance-based: Remove minutiae too close to each other (likely noise).
3. Angular consistency: Check if minutia angle aligns with local ridge orientation.
4. Neighborhood quality: Verify surrounding skeleton structure is consistent.

Parameters

- **minutiae** (*List[Minutia]*) – List of extracted minutiae (from extract_minutiae).
- **skeleton** (*np.ndarray*) – Thinned ridge skeleton used for context validation.
- **mask** (*np.ndarray*) – Boolean foreground mask.
- **validate_with_context** (*bool, optional*) – If True, apply neighborhood and angular consistency checks. Defaults to True.

Returns

Filtered list of valid minutiae with improved quality estimates.

Return type

List[*Minutia*]

Note

Removes minutiae pairs closer than MINUTIA_PAIR_DISTANCE (default 12 pixels). Angular consistency checks use 45 degrees threshold.

extractor.**zhang_suen_thinning**(*binary*, *max_iter=None*)

Apply Zhang-Suen thinning algorithm to obtain 1-pixel wide skeleton.

Parameters

- **binary** (*np.ndarray*) – Binary ridge image (uint8, 0 or 1).
- **max_iter** (*Optional[int], optional*) – Maximum iterations. Uses config default if None.

Returns

Thinned skeleton (uint8, 0 or 1).

Return type
np.ndarray

2.3 matching module

Matching Module for Supermatcher v1.0 (Hybrid)

This module contains:

- compute_geometric_minutiae_score: RANSAC-based minutiae matching
- FingerprintMatcher: Two-stage identification (hash + geometric reranking) and verification
- identify: 1:N identification with adaptive thresholds
- verify: 1:1 verification

`class matching.FingerprintMatcher(`*templates*, *hasher*`)`

Bases: `object`

Two-stage fingerprint matcher: hash-based + geometric reranking.

Implements production-grade identification and verification:

- Stage 1 (fast): Hash-based similarity (all candidates)
- Stage 2 (accurate): Geometric minutiae matching (top candidates with close scores)

`templates`

List of gallery templates.

Type
List[*FingerprintTemplate*]

`hasher`

CancelableHasher instance for protected template comparison.

Type
CancelableHasher

`adaptive_threshold(`*probe_quality*, *candidate_quality*`)`

Compute adaptive matching threshold based on template qualities.

Strategy (tuned for production):

- Both high quality (>0.7): Use standard threshold (0.78)
- Both medium quality (0.5): Relax slightly (-0.03 → 0.75)
- At least one low quality: Relax more (-0.05 → 0.73)

Parameters

- **probe_quality** (*float*) – Probe template quality [0, 1].
- **candidate_quality** (*float*) – Candidate template quality [0, 1].

Returns
Adjusted matching threshold.

Return type
float

`identify`(*probe*, *top_k=5*, *use_geometric_reranking=True*, *min_probe_quality=0.35*)

Two-stage identification: hash-based matching + geometric verification.

Algorithm:

1. Compute hash similarity for all gallery templates (Stage 1)
2. If top scores are ambiguous (diff < 0.02), apply geometric reranking (Stage 2)
3. Include tied candidates (score diff < 0.005) in reranking pool
4. Rerank top-N using combined score: 60% hash + 40% geometric
5. Return top-K matches with combined scores (reranked) or normalized scores (others)

Parameters

- **`probe`** (`FingerprintTemplate`) – Probe fingerprint template.
- **`top_k`** (*`int, optional`*) – Number of top matches to return. Defaults to 5.
- **`use_geometric_reranking`** (*`bool, optional`*) – Enable geometric reranking. Defaults to True.
- **`min_probe_quality`** (*`float, optional`*) – Minimum acceptable probe quality. Defaults to 0.35.

Returns

List of (identifier, score) tuples sorted by score (descending).

- Reranked candidates: combined score (60% hash + 40% geometric)
- Other candidates: normalized score (60% hash)

Return type

List[Tuple[str, float]]

Raises

`ValueError` – If probe quality is below min_probe_quality threshold.

`verify`(*probe*, *claimed_id*, *use_adaptive_threshold=True*)

1:1 verification: Check if probe matches claimed identity.

Compares probe against all templates with claimed_id, returns best match score.

Parameters

- **`probe`** (`FingerprintTemplate`) – Probe fingerprint template.
- **`claimed_id`** (*`str`*) – Claimed user identifier.
- **`use_adaptive_threshold`** (*`bool, optional`*) – Use quality-adaptive threshold. Defaults to True.

Returns

Tuple of (is_match, best_score):

- is_match: True if best_score threshold.
- best_score: Highest similarity score among claimed_id templates.

Return type

Tuple[bool, float]

Raises

`ValueError` – If no templates exist for claimed_id.

`matching.compute_geometric_minutiae_score`(*probe_minutiae*, *candidate_minutiae*, *distance_threshold=None*, *angle_threshold=None*)

Compute geometric similarity score between two minutiae sets using RANSAC alignment.

This function performs direct minutiae-to-minutiae matching using spatial and angular correspondence after rigid transformation alignment (rotation + translation).

Algorithm:

1. RANSAC to find best rigid transformation (rotation + translation)
2. Transform probe minutiae to candidate coordinate system
3. Count inliers (minutiae pairs within distance and angle thresholds)
4. Normalize score by larger set size (penalizes size mismatch)

Parameters

- **probe_minutiae** (*Sequence[Minutia]*) – Probe minutiae list.
- **candidate_minutiae** (*Sequence[Minutia]*) – Candidate minutiae list.
- **distance_threshold** (*Optional[float], optional*) – Max distance for correspondence. Uses config default if None.
- **angle_threshold** (*Optional[float], optional*) – Max angle difference for correspondence in radians. Uses config default if None.

Returns

Geometric similarity score in [0, 1] range.

Return type

float

2.4 models module

Data Structures for Supermatcher v1.0 (Hybrid)

This module defines the core data classes used throughout the fingerprint matching system. These classes are shared across all modules (preprocessing, extractor, template_creation, matching).

class `models.FingerprintTemplate`(*identifier*, *image_path*, *protected*, *bit_length*, *quality=0.0*, *raw_features=None*, *minutiae=None*, *fused=False*, *source_count=1*, *consensus_score=1.0*)

Bases: `object`

Complete fingerprint template with protected and raw features.

This is the main data structure for storing processed fingerprint data. It contains both the cancelable (protected) template and optional raw features for geometric matching and fusion.

identifier

Unique identifier (e.g., user ID).

Type

str

`image_path`

Path to original fingerprint image.

Type

Path

`protected`

Packed cancelable template (numpy array of bits).

Type

np.ndarray

`bit_length`

Length of protected template in bits.

Type

int

`quality`

Overall quality score [0.0, 1.0]. Defaults to 0.0.

Type

float

`raw_features`

Optional raw feature vector (for geometric matching).

Type

Optional[np.ndarray]

`minutiae`

Optional list of extracted minutiae.

Type

Optional[List[*Minutia*]]

`fused`

Whether this is a fused master template. Defaults to False.

Type

bool

`source_count`

Number of samples used to create this template. Defaults to 1.

Type

int

`consensus_score`

Consensus quality for fused templates [0.0, 1.0]. Defaults to 1.0.

Type

float

`bit_length: int`

`consensus_score: float = 1.0`

`fused: bool = False`

`identifier: str`

`image_path: Path`

`minutiae: Optional[List[Minutia]] = None`

`protected: ndarray`

`quality: float = 0.0`

`raw_features: Optional[ndarray] = None`

`source_count: int = 1`

class models.**FusionSettings**(*enabled=True*, *distance=12.0*, *angle_deg=15.0*, *min_consensus=0.5*, *keep_raw=False*, *mode='optimal'*)

Bases: `object`

Configuration for template fusion.

These settings control how multiple templates from the same user are fused into a single master template.

enabled

Whether fusion is enabled. Defaults to True.

Type
bool

distance

Spatial threshold for minutiae consensus (pixels). Defaults to 12.0.

Type
float

angle_deg

Angular threshold for minutiae consensus (degrees). Defaults to 15.0.

Type
float

min_consensus

Minimum consensus ratio [0.0, 1.0]. Defaults to 0.5.

Type
float

keep_raw

Whether to keep raw templates after fusion. Defaults to False.

Type
bool

mode

Fusion mode (optimal or other). Defaults to optimal.

Type
str

`angle_deg: float = 15.0`

`distance: float = 12.0`

`enabled: bool = True`

keep_raw: bool = False

min_consensus: float = 0.5

mode: str = 'optimal'

class models.**Minutia**(*x*, *y*, *angle*, *kind*, *quality*)

Bases: object

Fingerprint minutia (ridge ending or bifurcation).

x

X coordinate (pixels).

Type

float

y

Y coordinate (pixels).

Type

float

angle

Ridge direction in radians [0, 2].

Type

float

kind

Minutia type (ending or bifurcation).

Type

str

quality

Quality score [0.0, 1.0].

Type

float

angle: float

kind: str

quality: float

x: float

y: float

class models.**Pore**(*x*, *y*, *radius*, *strength*)

Bases: object

Level-3 feature: sweat pore.

x

X coordinate (pixels).

Type

float

`y`

Y coordinate (pixels).

Type
float

`radius`

Pore radius (pixels).

Type
float

`strength`

Detection strength/confidence [0.0, 1.0].

Type
float

`radius: float`

`strength: float`

`x: float`

`y: float`

2.5 models_serialization module

Secure Serialization for FingerprintTemplate (Protected + Minutiae only)

This module provides functions to serialize/deserialize fingerprint templates in a SECURE way: only the cancelable hash (protected) and minutiae are persisted.

RAW FEATURES ARE NEVER STORED to prevent template inversion attacks.

Migration from pickle (insecure) to dict-based serialization.

`models_serialization.`**`deserialize_template`**`(`*data*`)`

Deserialize template from secure dictionary.

Reconstructs a FingerprintTemplate with:

- Protected hash (restored from bytes)
- Minutiae (restored from list of dicts)
- raw_features = None (NEVER stored, can be recomputed if needed)

Parameters
`data` (*`Dict[str, Any]`*) – Dictionary with secure template data.

Returns
FingerprintTemplate object (with raw_features=None).

Return type
FingerprintTemplate

`models_serialization.`**`serialize_template`**`(`*template*`)`

Serialize template to secure dictionary (Protected + Minutiae only).

This function extracts ONLY the secure components of a template:

- Protected hash (cancelable biometric)
- Minutiae coordinates (reversible but limited information)
- Metadata (quality, fusion info, etc.)

RAW_FEATURES are NEVER included to prevent reconstruction attacks.

Parameters
`template` (`FingerprintTemplate`) – FingerprintTemplate object.

Returns
Dictionary with secure data only.

Return type
Dict[str, Any]

`models_serialization.`**`template_from_json`**(*json_str*)

Deserialize template from JSON string.

Parameters
`json_str` (*`str`*) – JSON string with template data.

Returns
FingerprintTemplate object.

Return type
FingerprintTemplate

`models_serialization.`**`template_from_secure_dict`**(*data*)

Deserialize template from secure dictionary.

Reconstructs a FingerprintTemplate with:

- Protected hash (restored from bytes)
- Minutiae (restored from list of dicts)
- raw_features = None (NEVER stored, can be recomputed if needed)

Parameters
`data` (*`Dict[str, Any]`*) – Dictionary with secure template data.

Returns
FingerprintTemplate object (with raw_features=None).

Return type
FingerprintTemplate

`models_serialization.`**`template_to_json`**(*template*)

Serialize template to JSON string (Protected as base64).

Useful for REST APIs and portable storage.

Parameters
`template` (`FingerprintTemplate`) – FingerprintTemplate object.

Returns
JSON string.

Return type
str

`models_serialization.`**`template_to_secure_dict`**`(`*template*`)`

Serialize template to secure dictionary (Protected + Minutiae only).

This function extracts ONLY the secure components of a template:

- Protected hash (cancelable biometric)
- Minutiae coordinates (reversible but limited information)
- Metadata (quality, fusion info, etc.)

RAW_FEATURES are NEVER included to prevent reconstruction attacks.

Parameters

`template` (`FingerprintTemplate`) – FingerprintTemplate object.

Returns

Dictionary with secure data only.

Return type

Dict[str, Any]

2.6 preprocessing module

Preprocessing Module for Supermatcher v1.0 (Hybrid)

This module contains all image preprocessing functions for fingerprint processing:

- Image loading and normalization
- Segmentation (foreground/background separation)
- Coherence-enhancing diffusion for ridge enhancement

Functions extracted from supermatcher_v0.5.1.py core pipeline.

`preprocessing.`**`block_variance_segmentation`**`(`*image*, *block_size=None*, *threshold=None*`)`

Segment fingerprint foreground from background using block variance.

Divides the image into blocks and computes variance for each block. High-variance blocks indicate ridge structures (foreground), while low-variance blocks indicate background or noise.

Parameters

- **`image`** (*`np.ndarray`*) – Input grayscale image (float32, 0-255).
- **`block_size`** (*`Optional[int], optional`*) – Size of square blocks. Uses config default if None.
- **`threshold`** (*`Optional[float], optional`*) – Variance threshold. Uses config default if None.

Returns

Boolean mask: True for foreground, False for background.

Return type

np.ndarray

Note

Applies morphological closing and opening to remove noise.

preprocessing.**coherence_diffusion**(*image*, *mask*, *iterations=None*, *dt=0.15*, *grad_sigma=1.0*, *tensor_sigma=2.0*, *alpha=0.01*, *beta=1.25*)

Apply coherence-enhancing diffusion (CED) for ridge enhancement.

Implements anisotropic diffusion based on the local structure tensor, which enhances ridge structures while preserving edges. Diffusion is stronger along ridge directions and weaker across ridges.

The algorithm:

1. Computes structure tensor from image gradients.
2. Calculates eigenvalues (1, 2) to determine local coherence.
3. Constructs diffusion tensor with anisotropic diffusivity.
4. Applies diffusion equation iteratively: I(t+1) = I(t) + dt * div(DůI)

Parameters

- **image** (*np.ndarray*) – Input grayscale image (float32, 0-255).
- **mask** (*np.ndarray*) – Boolean mask indicating valid foreground pixels.
- **iterations** (*Optional[int], optional*) – Number of diffusion iterations. Uses config default if None.
- **dt** (*float, optional*) – Time step size for numerical integration. Defaults to 0.15.
- **grad_sigma** (*float, optional*) – Sigma for Gaussian smoothing of gradients. Defaults to 1.0.
- **tensor_sigma** (*float, optional*) – Sigma for smoothing structure tensor. Defaults to 2.0.
- **alpha** (*float, optional*) – Minimum diffusivity perpendicular to ridges. Defaults to 0.01.
- **beta** (*float, optional*) – Maximum diffusivity parallel to ridges. Defaults to 1.25.

Returns

Enhanced image (float32, 0-255) with connected ridges.

Return type

np.ndarray

Note

Uses float64 precision internally for numerical stability. Coherence measure: exp(-(1-2)^2/(1ů2)) [0,1]

preprocessing.**gaussian_derivatives**(*image*, *sigma*)

Compute smoothed image gradients using Gaussian convolution.

Applies Gaussian blur followed by Sobel operators to compute stable gradients. Uses float64 precision for numerical stability.

Parameters

- **image** (*np.ndarray*) – Input grayscale image (any numeric type).
- **sigma** (*float*) – Standard deviation of Gaussian smoothing.

Returns
Tuple of (gx, gy) - gradient in x and y directions (float64).

Return type
Tuple[np.ndarray, np.ndarray]

`preprocessing.`**`load_grayscale_image`**(*path*)

Load a fingerprint image as grayscale float32.

Parameters
`path` (*`Path`*) – Path to the fingerprint image file.

Returns
Grayscale image as float32 (0-255 range).

Return type
np.ndarray

Raises
`FileNotFoundError` – If the image cannot be loaded.

`preprocessing.`**`normalise_image`**(*image*, *block_size=16*, *mean0=None*, *var0=None*)

Normalize image intensity using local block statistics.

Applies local normalization to compensate for uneven illumination and contrast. Each pixel is normalized based on the mean and variance of its local neighborhood.

Parameters

- **`image`** (*`np.ndarray`*) – Input grayscale image (any numeric type).
- **`block_size`** (*`int, optional`*) – Size of local neighborhood block. Defaults to 16.
- **`mean0`** (*`Optional[float], optional`*) – Target mean intensity. Uses config default if None.
- **`var0`** (*`Optional[float], optional`*) – Target variance. Uses config default if None.

Returns
Normalized image (float32, 0-255 range).

Return type
np.ndarray

Note

Formula: normalized = mean0 + (image - local_mean) * sqrt(var0 / local_var)

2.7 supermatcher_v1_0 module

Supermatcher v1.0 - Hybrid Fingerprint Identification/Authentication Pipeline

This module implements a modular, end-to-end pipeline for fingerprint identification and authentication.

ARCHITECTURE:

- **Modular design with 6 specialized modules:**
  - config.py: Centralized configuration
  - dataclasses.py: Core data structures

- preprocessing.py: Image processing (normalization, segmentation, diffusion)
- extractor.py: Feature extraction (orientation, frequency, minutiae, pores)
- template_creation.py: Hasher, fusion, quality assessment, I/O
- matching.py: Two-stage identification + verification

PIPELINE OVERVIEW:

1. Preprocessing: Normalization → Segmentation → Coherence Diffusion
2. Extraction: Orientation/Frequency → Log-Gabor Enhancement → Minutiae → Pores
3. Template Creation: Feature Vector → Quality Assessment → Protected Hash
4. Matching: Hash-based (Stage 1) → Geometric Reranking (Stage 2)

This script provides a command-line interface for enrolling, identifying, and verifying fingerprints using the above pipeline.

`class supermatcher_v1_0.`**`FingerprintPipeline`**(*hasher=None*, *include_level3=False*)

Bases: `object`

End-to-end fingerprint processing pipeline for biometric template creation.

This class processes a raw fingerprint image through all stages:

1. Preprocessing (load → normalize → segment → diffuse)
2. Feature extraction (orientation/frequency → Log-Gabor → minutiae → pores)
3. Template creation (feature vector → quality assessment → protected hash)

`hasher`

Instance for protected template generation.

Type

CancelableHasher

`include_level3`

Whether to extract Level-3 features (pores).

Type

bool

`process`(*image_path*, *identifier*, ** (Keyword-only parameters separator (PEP 3102))*, *verbose=False*)

Process a fingerprint image into a biometric template.

This method performs the complete pipeline: preprocessing, feature extraction, and template creation, resulting in a protected fingerprint template.

Parameters

- **`image_path`** (`Path`) – Path to the fingerprint image file.
- **`identifier`** (`str`) – User identifier for the template.
- **`verbose`** (`bool, optional`) – If True, print processing steps. Defaults to False.

Returns

The generated template with protected hash, minutiae, and quality score.

Return type

FingerprintTemplate

Raises

- **FileNotFoundError** – If the image_path does not exist.
- **ValueError** – If image processing fails.

supermatcher_v1_0.**enroll**(*image_paths*, *identifier*, *output_dir=None*, ***, *fusion_settings=None*, *quality_threshold=None*, *include_level3=False*, *verbose=True*)

Enroll a user by processing multiple fingerprint samples.

This function processes multiple fingerprint images for a single user, filters them by quality, optionally fuses multiple samples into a master template, and saves the templates to disk.

Workflow:

1. Process all images for the given identifier.
2. Filter by quality threshold.
3. Optionally fuse multiple samples into a master template.
4. Save templates to the output directory.

Parameters

- **image_paths** (*Sequence[Path]*) – List of fingerprint image paths for the same user.
- **identifier** (*str*) – User identifier (e.g., 101, john_doe).
- **output_dir** (*Optional[Path], optional*) – Directory to save templates. Uses default if None.
- **fusion_settings** (*Optional[FusionSettings], optional*) – Fusion configuration. None means no fusion.
- **quality_threshold** (*Optional[float], optional*) – Minimum quality to accept. Uses default if None.
- **include_level3** (*bool, optional*) – Extract Level-3 features. Defaults to False.
- **verbose** (*bool, optional*) – Print progress messages. Defaults to True.

Returns

List of FingerprintTemplate objects (fused + raw if keep_raw=True).

Return type

List[*FingerprintTemplate*]

supermatcher_v1_0.**enumerate_database**(*db_path*)

List all fingerprint images in a database directory.

Parameters

db_path (*Path*) – Path to the fingerprint database directory.

Returns

List of image file paths (sorted).

Return type

List[Path]

Raises

FileNotFoundError – If db_path doesnt exist or contains no images.

supermatcher_v1_0.**identify**(*probe_path*, *gallery_dir=None*, ***, *top_k=5*, *use_geometric_reranking=True*, *quality_threshold=None*, *prefer_fused=True*, *include_level3=False*, *verbose=True*)

Identify a probe fingerprint against a gallery (1:N matching).

This function performs identification by matching a probe fingerprint against all templates in the gallery using a two-stage matching process with optional geometric reranking.

Workflow:

1. Process the probe image.
2. Load gallery templates.
3. Use FingerprintMatcher.identify() with geometric reranking.
4. Return top-K matches.

Parameters

- **probe_path** (*Path*) – Path to the probe fingerprint image.
- **gallery_dir** (*Optional[Path], optional*) – Directory containing gallery templates. Uses default if None.
- **top_k** (*int, optional*) – Number of top matches to return. Defaults to 5.
- **use_geometric_reranking** (*bool, optional*) – Enable Stage 2 geometric matching. Defaults to True.
- **quality_threshold** (*Optional[float], optional*) – Minimum probe quality. Uses default if None.
- **prefer_fused** (*bool, optional*) – Prefer fused templates over raw. Defaults to True.
- **include_level3** (*bool, optional*) – Extract Level-3 features. Defaults to False.
- **verbose** (*bool, optional*) – Print matching details. Defaults to True.

Returns

List of (identifier, score) tuples sorted by score (descending).

Return type

List[Tuple[str, float]]

Raises

- **ValueError** – If probe quality is below threshold.
- **FileNotFoundError** – If gallery_dir contains no templates.

supermatcher_v1_0.**infer_identity_from_filename**(*path*)

Infer user identifier from filename (e.g., 101_1.tif → 101).

Parameters

path (*Path*) – Image file path.

Returns

User identifier (string before first underscore, or full stem if no underscore).

Return type

str

`supermatcher_v1_0.`**`set_cpu_affinity`**`()`

Set CPU affinity to P-cores only, if configured in the system and available.

This function restricts the process to run only on performance cores (P-cores), which can improve performance on hybrid CPU architectures.

`supermatcher_v1_0.`**`verify`**`(`*probe_path*, *claimed_id*, *gallery_dir=None*, *, *quality_threshold=None*, *use_adaptive_threshold=True*, *include_level3=False*, *verbose=True*`)`

Verify a probe fingerprint against a claimed identity (1:1 matching).

This function performs verification by matching a probe fingerprint against templates of a specific claimed identity using adaptive thresholding.

Workflow:

1. Process the probe image.
2. Load gallery templates for the claimed identity.
3. Use FingerprintMatcher.verify() with adaptive threshold.
4. Return (is_match, score).

Parameters

- **`probe_path`** (*`Path`*) – Path to the probe fingerprint image.
- **`claimed_id`** (*`str`*) – Claimed user identifier.
- **`gallery_dir`** (*`Optional[Path], optional`*) – Directory containing gallery templates. Uses default if None.
- **`quality_threshold`** (*`Optional[float], optional`*) – Minimum probe quality. Uses default if None.
- **`use_adaptive_threshold`** (*`bool, optional`*) – Use quality-based threshold. Defaults to True.
- **`include_level3`** (*`bool, optional`*) – Extract Level-3 features. Defaults to False.
- **`verbose`** (*`bool, optional`*) – Print verification details. Defaults to True.

Returns

Tuple of (is_match, score):

- is_match: True if score threshold.
- score: Best similarity score.

Return type

Tuple[bool, float]

Raises

`ValueError` – If probe quality is below threshold or no templates for claimed_id.

2.8 template_creation module

Template Creation and Fusion Module for Supermatcher v1.0 (Hybrid)

This module provides classes and functions for creating, fusing, and managing biometric fingerprint templates. It includes:

- CancelableHasher: Generates protected templates using random projections and binarization.

- Feature vector fusion: Combines feature vectors from multiple samples using quality-weighted averaging.
- Minutiae fusion: Merges minutiae sets using spatial clustering and RANSAC-based alignment.
- Template quality assessment: Computes a quality score based on multiple fingerprint metrics.
- Template I/O: Functions for saving and loading templates from disk.

All core template fusion and encoding logic is extracted and improved from the original supermatcher_v0.5.1.py fusion pipeline.

class `template_creation.`**`CancelableHasher`**(*feature_dim*, *projection_dim*, *key*, *hash_count=1*)

Bases: `object`

Cancelable biometric template encoder using random projection and binarization.

This class implements a protected template encoding scheme:
hash = sign(Px + b), where P is a key-derived random matrix and b is a random bias.

Properties:

- Non-invertible: Cannot recover original features from the hash.
- Revocable: Changing the key produces a different template.
- Renewable: Multiple independent templates can be generated per user.
- Similarity-preserving: Hamming distance between hashes correlates with feature distance.

`feature_dim`

Input feature vector dimension (default 736).

Type
int

`projection_dim`

Output hash dimension before packing (default 512 bits).

Type
int

`hash_count`

Number of independent hashes (default 2).

Type
int

`bit_length`

Total number of bits (projection_dim Œ hash_count).

Type
int

`property bit_length: int`

`encode`(*features*)

Encodes a feature vector into a packed binary template using the cancelable hashing scheme.

Parameters
`features` (*`np.ndarray`*) – Input feature vector of shape (feature_dim,).

Returns
Packed binary template as a uint8 array.

Return type
np.ndarray

`similarity`(*packed_a*, *packed_b*)

Computes the similarity between two packed templates as 1 minus the normalized Hamming distance.

Parameters

- **packed_a** (*np.ndarray*) – First packed template.
- **packed_b** (*np.ndarray*) – Second packed template.

Returns
Similarity score in [0, 1].

Return type
float

`template_creation.`**`align_minutiae_sets`**(*reference_minutiae*, *target_minutiae*, *image_shape*)

Align target minutiae to reference using RANSAC.

Parameters

- **reference_minutiae** (`Sequence[Minutia]`) – Reference minutiae list
- **target_minutiae** (`Sequence[Minutia]`) – Target minutiae to transform
- **image_shape** (`Tuple[int, int]`) – Image dimensions (height, width)

Return type
`Tuple[List[Minutia], float]`

Returns
Tuple of (aligned_minutiae, alignment_confidence)

`template_creation.`**`build_feature_vector`**(*minutiae*, *mask*, *orientation*, *frequency*, *skeleton*, *enhanced*, *diffused*, ***, *level3=None*, *use_consensus_weighting=True*)

Build feature vector from fingerprint processing outputs.

Feature components (total 736 dimensions): 1. Minutiae features (TOP-130): position, angle, quality Œ 130 minutiae = 650 dims 2. Global statistics: minutiae count, mask coverage, skeleton density 3. Minutiae statistics: quality, spatial distribution 4. Orientation histogram + statistics 5. Frequency histogram + statistics 6. Level-3 features (if available) 7. Intensity statistics (enhanced + diffused) 8. Gabor filter responses

Parameters

- **minutiae** (`Sequence[Minutia]`) – List of extracted minutiae
- **mask** (`ndarray`) – Processing outputs
- **orientation** (`ndarray`) – Processing outputs
- **frequency** (`ndarray`) – Processing outputs
- **skeleton** (`ndarray`) – Processing outputs
- **enhanced** (`ndarray`) – Processing outputs
- **diffused** (`ndarray`) – Processing outputs
- **level3** (`Optional[Sequence[Pore]]`) – Optional pore features
- **use_consensus_weighting** (`bool`) – If True, weight minutiae by consensus (for fused templates)

Return type
`ndarray`

Returns
Feature vector (numpy array, length FEATURE_VECTOR_DIM=736)

`template_creation.`**`create_fused_template`**`(`*identifier*, *templates*, *hasher*, *settings*`)`

Create fused template from multiple samples of same identity.

Parameters

- **`identifier`** (`str`) – User identifier
- **`templates`** (`Sequence[FingerprintTemplate]`) – Sequence of templates to fuse
- **`hasher`** (*`CancelableHasher`*) – CancelableHasher for protected template encoding
- **`settings`** (`FusionSettings`) – Fusion configuration

Return type
`Optional[FingerprintTemplate]`

Returns
Fused FingerprintTemplate or None if fusion fails

`template_creation.`**`estimate_rigid_transform_ransac`**`(`*src_points*, *dst_points*, *max_iterations=None*, *threshold=None*`)`

Estimate rigid transformation using RANSAC.

Parameters

- **`src_points`** (`ndarray`) – Source points (N, 2)
- **`dst_points`** (`ndarray`) – Destination points (N, 2)
- **`max_iterations`** (`int`) – Max RANSAC iterations (uses config default if None)
- **`threshold`** (`float`) – Inlier distance threshold (uses config default if None)

Return type
`Tuple[Optional[ndarray], float]`

Returns
Tuple of (transformation_matrix 2Œ3, confidence_score)

`template_creation.`**`evaluate_quality`**`(`*minutiae*, *mask*, *orientation*, *frequency*, *skeleton*, *enhanced*, *diffused*, ***, *level3=None*`)`

Estimate fingerprint quality score in [0, 1].

Combines multiple metrics: - Minutiae count/density - Foreground area ratio - Skeleton quality - Orientation/frequency validity - Contrast (enhanced and diffused) - Level-3 features (if available)

Return type
`float`

Returns
Quality score in [0, 1]

`template_creation.`**`fuse_feature_vectors`**`(`*templates*`)`

Fuse feature vectors using quality-weighted averaging.

Parameters
`templates` (`Sequence[FingerprintTemplate]`) – Sequence of FingerprintTemplate objects with raw_features

Return type
`Optional[ndarray]`

Returns
Fused feature vector (L2-normalized), or None if no valid vectors

`template_creation.`**`fuse_identity_templates`**`(`*templates*, *hasher*, *settings*`)`

Group templates by identifier and fuse each identitys templates.

Parameters

- **`templates`** (`Sequence[FingerprintTemplate]`) – Sequence of all templates
- **`hasher`** (*`CancelableHasher`*) – CancelableHasher instance
- **`settings`** (`FusionSettings`) – Fusion configuration

Return type
`List[FingerprintTemplate]`

Returns
List of fused templates (one per identity)

`template_creation.`**`fuse_minutiae_consensus`**`(`*templates*, *settings*`)`

Fuse minutiae from multiple templates using spatial clustering.

Parameters

- **`templates`** (`Sequence[FingerprintTemplate]`) – Sequence of FingerprintTemplate objects with minutiae
- **`settings`** (`FusionSettings`) – Fusion configuration (distance, angle_rad, min_consensus)

Return type
`List[Minutia]`

Returns
List of fused minutiae with consensus quality scores

`template_creation.`**`fuse_protected_templates`**`(`*templates*, *hasher*`)`

Fuse protected templates using majority voting on bits.

Parameters

- **`templates`** (`Sequence[FingerprintTemplate]`) – Sequence of FingerprintTemplate with protected hashes
- **`hasher`** (*`CancelableHasher`*) – CancelableHasher instance for bit length

Return type
`Tuple[Optional[ndarray], Optional[ndarray]]`

Returns
Tuple of (fused_bits, tie_mask) or (None, None)

`template_creation.`**`load_templates_from_directory`**`(`*directory*, *, *prefer_fused=True*, *include_raw_when_fused=False*`)`

Load templates from directory.

Parameters

- **`directory`** (`Path`) – Directory containing .fpt files
- **`prefer_fused`** (`bool`) – If True, prefer fused templates over raw

- **`include_raw_when_fused`** (`bool`) – If True, include raw templates even when fused exist

Return type

`List[FingerprintTemplate]`

Returns

List of FingerprintTemplate objects

`template_creation.`**`save_template`**(*template*, *output_dir*, ***, *overwrite=False*)

Save template to disk using pickle.

Return type

`Path`

`template_creation.`**`template_output_path`**(*output_dir*, *image_path*)

Generate output path for template file.

Return type

`Path`

2.9 webserver package

2.9.1 Subpackages

webserver.routes package

Submodules

webserver.routes.admin_routes module

Admin Routes System administration endpoints (stats, jobs, folder loading).

`async` `webserver.routes.admin_routes.`**`cancel_job`**(*job_id*)

Cancel a running job.

Path Parameters:

- job_id: Job ID (UUID)

Returns

Success/failure message - cancelled: True if cancelled, False if not running

Return type

- message

`async` `webserver.routes.admin_routes.`**`cleanup_jobs`**(*days=Form(7)*)

Clean up old completed/failed jobs.

Form Data:

- days: Age threshold in days (default 7)

Returns

Success message - removed: Number of jobs removed

Return type

- message

async webserver.routes.admin_routes.**get_audit_log**(*limit=100*)

Get audit log entries.

Query Parameters:

- limit: Maximum number of entries (default 100)

Returns
List of audit log entries

Return type

- logs

async webserver.routes.admin_routes.**get_job**(*job_id*)

Get job status and results.

Path Parameters:

- job_id: Job ID (UUID)

Returns
Job object with status, result, error, etc.

async webserver.routes.admin_routes.**get_stats**()

Get system statistics.

Returns
Number of enrolled users - avg_quality: Average template quality - encrypted: Whether database is encrypted - cached_templates: Number of templates in cache - running_jobs: Number of running background jobs

Return type

- num_users

async webserver.routes.admin_routes.**list_running_jobs**()

List all currently running jobs.

Returns
Dict mapping job_id -> job_info

Return type

- jobs

async webserver.routes.admin_routes.**load_folder**(*folder_path=Form(PydanticUndefined)*)

Start a background job to load fingerprints from folder.

Expected folder structure:

folder_path/

user1/
img1.png img2.png

user2/
img1.png img2.png

Form Data:

- folder_path: Path to folder containing user subfolders

Returns
Background job ID (UUID) - message: Info message

Return type

- job_id

`webserver.routes.admin_routes.set_job_manager(jm)`

Set global job manager reference.

webserver.routes.auth_routes module

Authentication Routes Login, token refresh, logout endpoints.

`class webserver.routes.auth_routes.LoginRequest(**data)`

Bases: `BaseModel`

Login request model.

`login: str`

`model_config: ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

`password: str`

`class webserver.routes.auth_routes.LoginResponse(**data)`

Bases: `BaseModel`

Login response model.

`expires_in: int`

`login: str`

`model_config: ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

`privilege: str`

`token: str`

`async webserver.routes.auth_routes.login(request, req)`

Authenticate user and return JWT token.

Request:

- login: Username
- password: Password

Response:

- token: JWT token
- login: Username
- privilege: User privilege (admin/endpoint)
- expires_in: Token expiry in seconds

async webserver.routes.auth_routes.**logout**(*request*, *user=Depends(dependency=<function get_current_user>*, *use_cache=True*, *scope=None)*)

Logout user (for logging purposes - JWT is stateless).

Note: Since JWT is stateless, this just logs the event. Client should discard the token.

async webserver.routes.auth_routes.**refresh_token**(*request*, *user=Depends(dependency=<function get_current_user>*, *use_cache=True*, *scope=None)*)

Refresh JWT token (extend expiry).

Requires valid JWT token in Authorization header.

Response:

- token: New JWT token with extended expiry

webserver.routes.biometric_routes module

Biometric Operations Routes Verify (1:1) and Identify (1:N) operations.

async webserver.routes.biometric_routes.**identify**(*image=File(PydanticUndefined)*, *top_k=Form(5)*)

1:N identification against all enrolled users.

Form Data:

- image: Probe fingerprint image
- top_k: Number of top matches to return (default 5)

Returns

True if match found above threshold - best_match_id: User ID of best match (if identified) - best_score: Score of best match - matches: List of top matches [{user_id, score, num_matched}] - total_users: Total number of enrolled users - threshold: Identification threshold used

Return type

- identified

webserver.routes.biometric_routes.**set_globals**(*pool*, *cache*)

Set global process pool and template cache references.

async webserver.routes.biometric_routes.**verify**(*user_id=Form(PydanticUndefined)*, *image=File(PydanticUndefined)*)

1:1 verification against enrolled user.

Form Data:

- user_id: User to verify against
- image: Probe fingerprint image

Returns

True if verified, False otherwise - score: Matching score - user_id: Claimed user ID - threshold: Verification threshold used

Return type

- match

webserver.routes.user_routes module

User Management Routes CRUD operations for fingerprint users.

async webserver.routes.user_routes.**add_user**(*user_id=Form(PydanticUndefined)*, *name=Form(PydanticUndefined)*, *images=File(PydanticUndefined)*)

Enroll a new user with fingerprint images.

Form Data:

- user_id: User identifier
- name: User name
- images: List of fingerprint images (PNG, JPG, TIF, etc.)

Returns

Enrolled user ID - quality: Template quality score - num_images: Number of images processed

Return type

- user_id

async webserver.routes.user_routes.**delete_all_users**()

Delete all users (admin only).

Returns

Success message - count: Number of users deleted

Return type

- message

async webserver.routes.user_routes.**delete_user**(*user_id*)

Delete a user.

Path Parameters:

- user_id: User identifier

Returns

Success message - user_id: Deleted user ID

Return type

- message

async webserver.routes.user_routes.**get_user**(*user_id*)

Get user details by ID.

Path Parameters:

- user_id: User identifier

Returns

User object (without template data)

async webserver.routes.user_routes.**list_users**()

List all enrolled users.

Returns

List of user objects (without template data)

Return type

- users

webserver.routes.user_routes.**set_globals**(*pool*, *cache*)

Set global process pool and template cache references.

async webserver.routes.user_routes.**update_user**(*user_id*, *name=Form(None)*, *images=File(None)*)

Update user details and/or re-enroll with new images.

Path Parameters:

- user_id: User identifier

Form Data (all optional):

- name: New user name
- images: New fingerprint images (triggers re-enrollment)

Returns

Success message - user_id: Updated user ID

Return type

- message

Module contents

Routes package - API endpoint modules

2.9.2 Submodules

2.9.3 webserver.auth module

Webserver Authentication and Authorization

This module provides JWT token management, rate limiting, password hashing, and FastAPI security dependencies for the biometric webserver. It supports configurable token expiry, per-operation rate limiting, privilege enforcement, and secure password storage. Utility functions are included for extracting client IPs and cleaning up rate limit storage.

Features:

- JWT token creation and validation with configurable expiry
- Rate limiting per operation type (login, verify, identify, etc.)
- FastAPI dependency decorators for authentication and privilege enforcement
- IP-based and user-based rate limiting
- Secure password hashing and verification (bcrypt)

webserver.auth.**check_rate_limit**(*identifier*, *operation*, *max_requests=None*, *window_seconds=None*)

Check if an operation is within rate limits.

Parameters

- **identifier** (str) – Unique identifier (IP address, username, etc.)
- **operation** (str) – Operation type (must be in RATE_LIMITS or custom)
- **max_requests** (Optional[int]) – Override max requests (uses RATE_LIMITS if None)
- **window_seconds** (Optional[int]) – Override time window (uses RATE_LIMITS if None)

Returns

bool, retry_after: Optional[int]) - allowed: True if within limit, False if exceeded - retry_after: Seconds until next allowed request (if exceeded)

Return type

Tuple of (allowed

Example

```
>>> allowed, retry = check_rate_limit("192.168.1.100", "login")
>>> if not allowed:
...     raise HTTPException(429, f"Rate limit exceeded. Retry after {retry}s")
```

webserver.auth.**cleanup_rate_limit_storage**()

Clean up expired rate limit entries (call periodically).

Removes all entries older than the longest rate limit window.

webserver.auth.**create_token**(*user_data*, *privilege=None*)

Create a JWT token for authenticated user.

Parameters

- **user_data** (Dict[str, str]) – User information dict (must have login and privilege keys)
- **privilege** (Optional[str]) – Override privilege (optional, uses user_data[privilege] by default)

Return type

str

Returns

JWT token string

Example

```
>>> token = create_token({"login": "admin", "privilege": "admin"})
>>> print(token[:20])
eyJhbGciOiJIUzI1NiIs...
```

webserver.auth.**get_client_ip**(*request*)

Extract client IP address from request (handles proxies).

Parameters

request (Request) – FastAPI request object

Return type

str

Returns

Client IP address string

`async webserver.auth.`**`get_current_user`**`(`*request*, *credentials=Depends(dependency=<fastapi.security.http.HTTPBearer object>*, *use_cache=True*, *scope=None)*`)`

FastAPI dependency: Extract and verify JWT token from Authorization header.

Parameters

- **`request`** (`Request`) – FastAPI request object
- **`credentials`** (`HTTPAuthorizationCredentials`) – HTTP Bearer credentials

Return type

`Dict[str, Any]`

Returns

Decoded token payload (user data)

Raises

`HTTPException` – If token is missing or invalid (401)

Usage:

@app.get(/protected) async def protected_route(user = Depends(get_current_user)):

return {message: fHello {user[login]}}

`webserver.auth.`**`hash_password`**`(`*password*`)`

Hash a password using bcrypt.

Parameters

`password` (`str`) – Plain text password

Return type

`str`

Returns

Bcrypt hash string

Example

```
>>> hashed = hash_password("secret123")
>>> print(hashed[:7])
$2b$12$
```

`webserver.auth.`**`is_ip_locked`**`(`*ip*`)`

Check if an IP is locked out due to failed login attempts.

Parameters

`ip` (`str`) – Client IP address

Returns

bool, retry_after: Optional[int]) - locked: True if IP is locked out - retry_after: Seconds until unlock (if locked)

Return type

Tuple of (locked

`webserver.auth.`**`rate_limit`**`(`*operation*`)`

FastAPI dependency factory: Enforce rate limiting for an operation.

Parameters

`operation` (`str`) – Operation type (must be in RATE_LIMITS or defaults to 10/min)

Returns
FastAPI dependency function

Raises
`HTTPException` – If rate limit exceeded (429)

Usage:
@app.post(/api/verify, dependencies=[Depends(rate_limit(verify))]) async def verify_fingerprint():

`webserver.auth.`**`record_login_attempt`**(*ip*, *success*)

Record a login attempt (for account lockout protection).

Parameters

- **`ip`** (`str`) – Client IP address
- **`success`** (`bool`) – Whether login was successful

Notes

- Failed attempts are tracked for MAX_LOGIN_ATTEMPTS lockout
- Successful logins clear the history

`webserver.auth.`**`refresh_token`**(*old_token*)

Refresh an existing token (extends expiry).

Parameters
`old_token` (`str`) – Current JWT token

Return type
`str`

Returns
New JWT token with extended expiry

Raises
`HTTPException` – If token is invalid (401)

`webserver.auth.`**`require_auth`**(*required_privilege=None*)

FastAPI dependency factory: Require authentication with optional privilege check.

Parameters
`required_privilege` (`Optional[str]`) – Minimum required privilege (PRIVILEGE_ADMIN or PRIVILEGE_ENDPOINT) If None, any authenticated user is allowed

Returns
FastAPI dependency function

Raises
`HTTPException` – If unauthorized (401) or insufficient privilege (403)

Usage:
# Any authenticated user @app.get(/users, dependencies=[Depends(require_auth())])

# Admin only @app.delete(/users/{user_id}, dependencies=[Depends(require_auth(PRIVILEGE_ADMIN))])

webserver.auth.**verify_password**(*password*, *hashed*)

Verify a password against a bcrypt hash.

Parameters

- **password** (str) – Plain text password
- **hashed** (str) – Bcrypt hash string

Return type

bool

Returns

True if password matches, False otherwise

Example

```
>>> hashed = hash_password("secret123")
>>> verify_password("secret123", hashed)
True
>>> verify_password("wrong", hashed)
False
```

webserver.auth.**verify_token**(*token*)

Verify and decode a JWT token.

Parameters

token (str) – JWT token string

Return type

Dict[str, Any]

Returns

Decoded payload dict with login and privilege

Raises

HTTPException – If token is invalid or expired (401)

Example

```
>>> payload = verify_token("eyJhbGci...")
>>> print(payload['login'])
admin
```

2.9.4 webserver.biometric_worker module

Biometric Worker Functions for ProcessPool

This module provides isolated worker functions for CPU-bound biometric operations, designed to run in separate processes via ProcessPoolExecutor. Functions include 1:1 verification, 1:N identification, user enrollment, and batch folder loading. All heavy imports (e.g., supermatcher) are performed inside the worker functions to avoid pickling issues and maximize multiprocessing efficiency.

Multiprocessing Strategy:

1. VERIFY (1:1 matching): No multiprocessing within a single request; server-level parallelism only.
2. IDENTIFY (1:N matching): Adaptive multiprocessing; sequential for small galleries, parallel for large galleries (20 users).

3. ADD_USER (enrollment): No multiprocessing within a single user; server-level parallelism for multiple requests.
4. LOAD_FOLDER (batch enrollment): Full multiprocessing; each user enrolled in parallel.

`webserver.biometric_worker.`**`worker_enroll`**`(`*image_paths*, *user_id*, *settings*`)`

Enroll a user from a list of fingerprint images (runs in ProcessPool).

Parameters

- **`image_paths`** (*`List[str]`*) – List of image file paths.
- **`user_id`** (*`str`*) – User identifier.
- **`settings`** (*`dict`*) – Enrollment settings, including quality_threshold and fusion.

Returns

Enrollment result with keys:

- success (bool): Whether enrollment succeeded.
- template_secure (dict): Secure serialization of the template.
- quality (float): Quality score of the template.
- num_images (int): Number of images used.
- error (Optional[str]): Error message if enrollment failed.

Return type

dict

`webserver.biometric_worker.`**`worker_enroll_single_user`**`(`*user_id*, *image_paths*, *settings*`)`

Enroll a single user from multiple fingerprint images (for parallel processing).

Parameters

- **`user_id`** (*`str`*) – User identifier.
- **`image_paths`** (*`List[str]`*) – List of image file paths for this user.
- **`settings`** (*`dict`*) – Enrollment settings, including quality_threshold and fusion.

Returns

Enrollment result with keys:

- success (bool): Whether enrollment succeeded.
- user_id (str): User identifier.
- template_secure (dict): Secure serialization of the template.
- quality (float): Quality score of the template.
- num_images (int): Number of images used.
- error (Optional[str]): Error message if enrollment failed.

Return type

dict

`webserver.biometric_worker.`**`worker_identify`**`(`*probe_path*, *gallery_secure_dicts*, *threshold*, *settings*, *top_k=5*`)`

Perform 1:N fingerprint identification of a probe image against a gallery.

Parameters

- **probe_path** (*str*) – Path to the probe image file.
- **gallery_secure_dicts** (*Dict[str, Dict[str, Any]]*) – Mapping of user_id to secure dict of FingerprintTemplate.
- **threshold** (*float*) – Identification threshold for a match.
- **settings** (*dict*) – Matching settings.
- **top_k** (*int, optional*) – Number of top matches to return. Defaults to 5.

Returns

Identification result with keys:

- success (bool): Whether identification succeeded.
- matches (List[dict]): List of matches with user_id, score, and num_matched.
- identified (bool): Whether a match above threshold was found.
- best_match_id (Optional[str]): User ID of the best match, if any.
- best_score (float): Score of the best match.
- error (Optional[str]): Error message if identification failed.

Return type
dict

`webserver.biometric_worker.`**`worker_load_folder`**(*folder_path*, *settings*, *progress_callback=None*)

Batch-enroll all users from a folder structure using parallel processing.

Expects folder structure:

folder_path/

user1/
img1.png img2.png

user2/
img1.png img2.png

Parameters

- **folder_path** (*str*) – Path to the folder containing user subfolders or flat image files.
- **settings** (*dict*) – Enrollment settings.
- **progress_callback** (*Optional[Callable]*) – Optional callback function called as progress is made.

Returns

Batch enrollment result with keys:

- success (bool): Whether the operation succeeded.
- enrolled (List[dict]): List of successfully enrolled users with user_id, template_secure, and quality.
- failed (List[dict]): List of failed enrollments with user_id and error.
- total (int): Total number of users processed.
- error (Optional[str]): Error message if the operation failed.

Return type
dict

`webserver.biometric_worker.`**`worker_match_probe_against_gallery_chunk`**(*probe_secure_dict*, *gallery_chunk_secure_dicts*, *hasher_params*)

Match a probe fingerprint against a chunk of gallery templates in parallel.

Used for parallel identification in large galleries. Each chunk is processed in a separate process.

Parameters

- **`probe_secure_dict`** (*`Dict[str, Any]`*) – Secure dict of the probe FingerprintTemplate.
- **`gallery_chunk_secure_dicts`** (*`Dict[str, Dict[str, Any]]`*) – Mapping of user_id to secure dict for a subset of the gallery.
- **`hasher_params`** (*`dict`*) – Parameters for the CancelableHasher.

Returns
List of (user_id, score) tuples for this chunk.

Return type
List[Tuple[str, float]]

`webserver.biometric_worker.`**`worker_verify`**(*probe_path*, *template_secure_dict*, *threshold*, *settings*)

Perform 1:1 fingerprint verification between a probe image and an enrolled template.

Parameters

- **`probe_path`** (*`str`*) – Path to the probe image file.
- **`template_secure_dict`** (*`Dict[str, Any]`*) – Secure dict of the enrolled FingerprintTemplate.
- **`threshold`** (*`float`*) – Verification threshold for a match.
- **`settings`** (*`dict`*) – Matching settings.

Returns

Verification result with keys:

- success (bool): Whether verification succeeded.
- match (bool): Whether the probe matches the template.
- score (float): Combined match score.
- num_matched (int): Number of matched minutiae (approximate).
- error (Optional[str]): Error message if verification failed.

Return type
dict

2.9.5 webserver.config module

WebServer Configuration

Centralized configuration for the biometric webserver. All settings, including paths, server parameters, authentication, security, biometric thresholds, multiprocessing, background jobs, and cache, are defined here for easy adjustment without modifying the source code. Helper functions are provided for secure key management and directory setup.

webserver.config.**ensure_directories**()

Ensure all required directories for certificates, logs, and database exist.

Creates directories if they do not already exist.

webserver.config.**get_db_key**()

Get or generate the database encryption key.

If the key file exists, reads and returns the key. Otherwise, generates a new secure key, saves it to the file system, and returns it.

Returns
The database encryption key.

Return type
str

webserver.config.**get_jwt_secret**()

Get or generate the JWT secret key for token signing.

If the secret file exists, reads and returns the secret. Otherwise, generates a new secure secret, saves it to the file system, and returns it.

Returns
The JWT secret key.

Return type
str

2.9.6 webserver.database module

Database Module - SQLite Encrypted Database Management

This module handles all database operations for the biometric webserver, including user authentication, secure fingerprint template storage, audit logging, and background job tracking. It uses SQLCipher for encrypted storage when available, and stores fingerprint templates in a secure, non-pickle format (protected hash and minutiae only). All operations are logged for auditing and compliance.

Tables:

- auth_users: Authentication credentials (admin/endpoint)
- fingerprints: Biometric templates (secure format, NO pickle)
- audit_log: All operations log
- jobs: Background job status

Security:

- Templates are stored as protected hash and minutiae JSON (no raw features, no pickle blobs)
- All sensitive operations are audited

class webserver.database.**BiometricDatabase**(*db_path=None*, *encryption_key=None*)

Bases: object

SQLite database manager for the biometric system with encryption support.

This class provides methods for user authentication, secure fingerprint template storage, audit logging, and background job management. It supports encrypted storage using SQLCipher and ensures all sensitive operations are logged for security and compliance.

add_fingerprint(*user_id*, *name*, *template_obj*, *username='system'*)

Add a fingerprint template to the database in secure format.

Parameters

- **user_id** (*str*) – Unique user identifier.
- **name** (*str*) – Users name.
- **template_obj** (*Any*) – FingerprintTemplate object from supermatcher v1.0.
- **username** (*str, optional*) – Username who performed the operation. Defaults to system.

Returns

True if the template was added successfully, False if the user already exists.

Return type

bool

authenticate(*username*, *password*)

Authenticate a user by verifying their username and password.

Parameters

- **username** (*str*) – Username to authenticate.
- **password** (*str*) – Plain text password to verify.

Returns

Dictionary with user info if authentication succeeds, None otherwise.

Return type

Optional[Dict[str, str]]

cleanup_old_jobs(*hours=24*)

Delete completed or failed jobs older than a specified number of hours.

Parameters

hours (*int, optional*) – Delete jobs older than this many hours. Defaults to 24.

Returns

Number of deleted jobs.

Return type

int

close()

Close the database connection.

create_auth_user(*username*, *password*, *privilege*)

Create a new authentication user in the database.

Parameters

- **username** (*str*) – Username for the new user.
- **password** (*str*) – Plain text password (will be securely hashed).
- **privilege** (*str*) – User privilege, either admin or endpoint.

Returns

True if the user was created successfully, False if the username already exists.

Return type
bool

`create_job`(*job_id*, *job_type*, *created_by*)

Create a new background job entry in the database.

Parameters

- `job_id` (*str*) – Unique job identifier.
- `job_type` (*str*) – Type of job (e.g., load_folder).
- `created_by` (*str*) – Username who created the job.

Returns
True if the job was created successfully, False if the job ID already exists.

Return type
bool

`delete_all_fingerprints`(*username='system'*)

Delete all fingerprints from the database.

Parameters
`username` (*str, optional*) – Username who performed the operation. Defaults to system.

Returns
Number of deleted fingerprint records.

Return type
int

`delete_fingerprint`(*user_id*, *username='system'*)

Delete a fingerprint from the database by user ID.

Parameters

- `user_id` (*str*) – User identifier.
- `username` (*str, optional*) – Username who performed the operation. Defaults to system.

Returns
True if the fingerprint was deleted successfully, False otherwise.

Return type
bool

`get_all_templates`()

Retrieve all fingerprint templates for caching or batch operations.

Returns
List of (user_id, template_object) tuples.

Return type
List[Tuple[str, Any]]

`get_audit_log`(*limit=100*)

Retrieve recent audit log entries from the database.

Parameters
`limit` (*int, optional*) – Maximum number of entries to return. Defaults to 100.

Returns
List of audit log entry dictionaries.

Return type
List[Dict[str, Any]]

`get_fingerprint`(*user_id*)

Retrieve a fingerprint template and metadata for a given user ID.

Parameters
`user_id` (*str*) – User identifier.

Returns
Dictionary with template info and metadata, or None if not found.

Return type
Optional[Dict[str, Any]]

`get_job`(*job_id*)

Retrieve the status and details of a background job by job ID.

Parameters
`job_id` (*str*) – Job identifier.

Returns
Dictionary with job info if found, None otherwise.

Return type
Optional[Dict[str, Any]]

`get_stats`()

Get system statistics for users, templates, and authentication.

Returns
Dictionary with statistics (user count, average quality, etc.).

Return type
Dict[str, Any]

`list_fingerprints`()

List all fingerprints in the database (without template blobs).

Returns
List of dictionaries with fingerprint metadata.

Return type
List[Dict[str, Any]]

`reset_admin_password`(*new_password*)

Reset the admin password for account recovery.

Parameters
`new_password` (*str*) – New plain text password for the admin account.

Returns
True if the password was reset successfully, False otherwise.

Return type
bool

`update_fingerprint`(*user_id*, *name=None*, *template_obj=None*, *username='system'*)

Update a fingerprints metadata or template in the database.

Parameters

- `user_id` (*str*) – User identifier.
- `name` (*str, optional*) – New name for the user.
- `template_obj` (*Any, optional*) – New fingerprint template object.
- `username` (*str, optional*) – Username who performed the operation. Defaults to system.

Returns

True if the fingerprint was updated successfully, False otherwise.

Return type

bool

`update_job`(*job_id*, *status=None*, *progress=None*, *result=None*, *error=None*)

Update the status or details of a background job in the database.

Parameters

- `job_id` (*str*) – Job identifier.
- `status` (*str, optional*) – New status (pending, processing, completed, failed).
- `progress` (*str, optional*) – Progress information (JSON string).
- `result` (*str, optional*) – Result data (JSON string).
- `error` (*str, optional*) – Error message.

Returns

True if the job was updated successfully, False otherwise.

Return type

bool

2.9.7 webserver.jobs module

Background Job Management

This module manages long-running background tasks such as folder loading and batch operations, with database persistence and progress tracking. It provides a JobManager class for creating, tracking, and cleaning up jobs, and integrates with the biometric worker pool and logging system.

`class webserver.jobs.JobManager`(*db*, *executor*)

Bases: `object`

Manages background jobs with database persistence and progress tracking.

The JobManager handles creation, execution, status tracking, and cleanup of background jobs such as folder loading. It uses a ProcessPoolExecutor for running jobs asynchronously and updates job status in the database. Job progress and results are logged for auditing and monitoring.

`cancel_job`(*job_id*)

Attempt to cancel a running job.

Parameters

`job_id` (*str*) – Job ID to cancel.

Returns

True if the job was cancelled, False if it was not running or could not be cancelled.

Return type
bool

`cleanup_old_jobs`(*days=7*)

Remove old completed or failed jobs from the database.

Parameters
`days` (*int*) – Age threshold in days. Jobs older than this will be deleted.

Returns
Number of jobs removed from the database.

Return type
int

`async create_load_folder_job`(*folder_path*, *created_by*)

Create and start a background job to load a folder of user subfolders.

This method creates a new job record in the database, submits the folder loading task to the process pool, and tracks its progress asynchronously. The job status is updated throughout its lifecycle.

Parameters

- `folder_path` (*str*) – Path to the folder containing user subfolders.
- `created_by` (*str*) – Username who created the job.

Returns
Job ID (UUID) of the created job.

Return type
str

`get_job_status`(*job_id*)

Retrieve the status and details of a job from the database.

Parameters
`job_id` (*str*) – Job ID to query.

Returns
Job dictionary if found, otherwise None.

Return type
Optional[Dict[str, Any]]

`get_running_jobs`()

Get a dictionary of currently running jobs and their information.

Returns
Mapping of job_id to job information for all running jobs.

Return type
Dict[str, Any]

2.9.8 webserver.logger module

Webserver Logging System

This module provides a structured logging system for the webserver, supporting multiple log files with automatic rotation and convenience functions for logging access, authentication, biometric operations, errors, jobs, and server events. It also supports structured JSON logging for external aggregation and log cleanup utilities.

Log Files:

- access.log: HTTP requests (IP, endpoint, status, duration)
- auth.log: Authentication events (login, logout, failed attempts)
- biometric.log: Biometric operations (enroll, verify, identify results)
- error.log: Application errors and exceptions

`webserver.logger.`**`cleanup_old_logs`**`(`*days=30*`)`

Delete log files older than the specified number of days from the log directory.

Parameters
`days` (*`int`*) – Age threshold in days. Files older than this will be deleted.

`webserver.logger.`**`get_logger`**`(`*name*`)`

Get a custom logger that writes to error.log, for specialized logging needs.

Parameters
`name` (*`str`*) – Logger name.

Returns
Logger instance for custom usage.

Return type
logging.Logger

`webserver.logger.`**`log_access`**`(`*ip*, *method*, *endpoint*, *status_code*, *duration_ms*, *user=None*`)`

Log an HTTP access event to the access log.

Parameters

- **`ip`** (*`str`*) – Client IP address.
- **`method`** (*`str`*) – HTTP method (GET, POST, etc.).
- **`endpoint`** (*`str`*) – Request endpoint path.
- **`status_code`** (*`int`*) – HTTP status code returned.
- **`duration_ms`** (*`float`*) – Request duration in milliseconds.
- **`user`** (*`Optional[str]`*) – Authenticated username, if available.

`webserver.logger.`**`log_auth`**`(`*event*, *login*, *ip*, *success=True*, *details=None*`)`

Log an authentication event to the authentication log.

Parameters

- **`event`** (*`str`*) – Event type (e.g., LOGIN, LOGOUT, TOKEN_REFRESH).
- **`login`** (*`str`*) – Username involved in the event.
- **`ip`** (*`str`*) – Client IP address.
- **`success`** (*`bool`*) – Whether the authentication operation succeeded. Defaults to True.
- **`details`** (*`Optional[str]`*) – Additional details about the event.

`webserver.logger.`**`log_biometric`**`(`*operation*, *user_id*, *result*, *details=None*, *performed_by=None*`)`

Log a biometric operation event to the biometric log.

Parameters

- **`operation`** (*`str`*) – Operation type (ENROLL, VERIFY, IDENTIFY).
- **`user_id`** (*`Optional[str]`*) – Target user ID (for enroll/verify) or None (for identify).

- **result** (*str*) – Operation result (e.g., SUCCESS, FAILURE, NO_MATCH).
- **details** (*Optional[Dict[str, Any]]*) – Additional details such as score, quality, matches, etc.
- **performed_by** (*Optional[str]*) – Username who performed the operation.

`webserver.logger.`**`log_error`**(*error*, *context=None*, *user=None*, *ip=None*)

Log an application error or exception to the error log.

Parameters

- **error** (*Exception*) – Exception object to log.
- **context** (*Optional[str]*) – Context where the error occurred (e.g., endpoint, function name).
- **user** (*Optional[str]*) – Authenticated username, if available.
- **ip** (*Optional[str]*) – Client IP address, if available.

`webserver.logger.`**`log_job`**(*job_id*, *job_type*, *status*, *details=None*, *created_by=None*)

Log a background job event to the access log.

Parameters

- **job_id** (*str*) – Unique job identifier.
- **job_type** (*str*) – Type of job (e.g., LOAD_FOLDER).
- **status** (*str*) – Job status (CREATED, PROCESSING, COMPLETED, FAILED).
- **details** (*Optional[Dict[str, Any]]*) – Additional details about the job.
- **created_by** (*Optional[str]*) – Username who created the job.

`webserver.logger.`**`log_json`**(*logger_type*, *data*)

Log structured JSON data to the specified logger for external log aggregation.

Parameters

- **logger_type** (*str*) – Logger to use (access, auth, biometric, error).
- **data** (*Dict[str, Any]*) – Dictionary to log as JSON.

`webserver.logger.`**`log_shutdown`**()

Log server shutdown event to the access log.

`webserver.logger.`**`log_startup`**(*info*)

Log server startup information to the access log.

Parameters

info (*Dict[str, Any]*) – Startup information such as host, port, SSL, number of templates, etc.

2.9.9 webserver.server module

FastAPI WebServer - Main Application

Production-ready biometric authentication server with JWT, rate limiting, and ProcessPool.

2.9.10 Module contents

Webserver Package - Biometric Authentication System FastAPI-based REST API with JWT authentication and ProcessPool biometric operations.

class webserver.**BiometricDatabase**(*db_path=None*, *encryption_key=None*)

Bases: `object`

SQLite database manager for the biometric system with encryption support.

This class provides methods for user authentication, secure fingerprint template storage, audit logging, and background job management. It supports encrypted storage using SQLCipher and ensures all sensitive operations are logged for security and compliance.

add_fingerprint(*user_id*, *name*, *template_obj*, *username='system'*)

Add a fingerprint template to the database in secure format.

Parameters

- **user_id** (*str*) – Unique user identifier.
- **name** (*str*) – Users name.
- **template_obj** (*Any*) – FingerprintTemplate object from supermatcher v1.0.
- **username** (*str, optional*) – Username who performed the operation. Defaults to system.

Returns

True if the template was added successfully, False if the user already exists.

Return type

bool

authenticate(*username*, *password*)

Authenticate a user by verifying their username and password.

Parameters

- **username** (*str*) – Username to authenticate.
- **password** (*str*) – Plain text password to verify.

Returns

Dictionary with user info if authentication succeeds, None otherwise.

Return type

Optional[Dict[str, str]]

cleanup_old_jobs(*hours=24*)

Delete completed or failed jobs older than a specified number of hours.

Parameters

hours (*int, optional*) – Delete jobs older than this many hours. Defaults to 24.

Returns

Number of deleted jobs.

Return type

int

close()

Close the database connection.

`create_auth_user`(*username*, *password*, *privilege*)

Create a new authentication user in the database.

Parameters

- **username** (*str*) – Username for the new user.
- **password** (*str*) – Plain text password (will be securely hashed).
- **privilege** (*str*) – User privilege, either admin or endpoint.

Returns

True if the user was created successfully, False if the username already exists.

Return type

bool

`create_job`(*job_id*, *job_type*, *created_by*)

Create a new background job entry in the database.

Parameters

- **job_id** (*str*) – Unique job identifier.
- **job_type** (*str*) – Type of job (e.g., load_folder).
- **created_by** (*str*) – Username who created the job.

Returns

True if the job was created successfully, False if the job ID already exists.

Return type

bool

`delete_all_fingerprints`(*username='system'*)

Delete all fingerprints from the database.

Parameters

username (*str, optional*) – Username who performed the operation. Defaults to system.

Returns

Number of deleted fingerprint records.

Return type

int

`delete_fingerprint`(*user_id*, *username='system'*)

Delete a fingerprint from the database by user ID.

Parameters

- **user_id** (*str*) – User identifier.
- **username** (*str, optional*) – Username who performed the operation. Defaults to system.

Returns

True if the fingerprint was deleted successfully, False otherwise.

Return type

bool

`get_all_templates()`

Retrieve all fingerprint templates for caching or batch operations.

Returns
List of (user_id, template_object) tuples.

Return type
List[Tuple[str, Any]]

`get_audit_log(`*limit=100*`)`

Retrieve recent audit log entries from the database.

Parameters
limit (*int, optional*) – Maximum number of entries to return. Defaults to 100.

Returns
List of audit log entry dictionaries.

Return type
List[Dict[str, Any]]

`get_fingerprint(`*user_id*`)`

Retrieve a fingerprint template and metadata for a given user ID.

Parameters
user_id (*str*) – User identifier.

Returns
Dictionary with template info and metadata, or None if not found.

Return type
Optional[Dict[str, Any]]

`get_job(`*job_id*`)`

Retrieve the status and details of a background job by job ID.

Parameters
job_id (*str*) – Job identifier.

Returns
Dictionary with job info if found, None otherwise.

Return type
Optional[Dict[str, Any]]

`get_stats()`

Get system statistics for users, templates, and authentication.

Returns
Dictionary with statistics (user count, average quality, etc.).

Return type
Dict[str, Any]

`list_fingerprints()`

List all fingerprints in the database (without template blobs).

Returns
List of dictionaries with fingerprint metadata.

Return type
List[Dict[str, Any]]

`reset_admin_password`(*new_password*)

Reset the admin password for account recovery.

Parameters

`new_password` (*str*) – New plain text password for the admin account.

Returns

True if the password was reset successfully, False otherwise.

Return type

bool

`update_fingerprint`(*user_id*, *name=None*, *template_obj=None*, *username='system'*)

Update a fingerprints metadata or template in the database.

Parameters

- `user_id` (*str*) – User identifier.
- `name` (*str, optional*) – New name for the user.
- `template_obj` (*Any, optional*) – New fingerprint template object.
- `username` (*str, optional*) – Username who performed the operation. Defaults to system.

Returns

True if the fingerprint was updated successfully, False otherwise.

Return type

bool

`update_job`(*job_id*, *status=None*, *progress=None*, *result=None*, *error=None*)

Update the status or details of a background job in the database.

Parameters

- `job_id` (*str*) – Job identifier.
- `status` (*str, optional*) – New status (pending, processing, completed, failed).
- `progress` (*str, optional*) – Progress information (JSON string).
- `result` (*str, optional*) – Result data (JSON string).
- `error` (*str, optional*) – Error message.

Returns

True if the job was updated successfully, False otherwise.

Return type

bool

`webserver.create_token`(*user_data*, *privilege=None*)

Create a JWT token for authenticated user.

Parameters

- `user_data` (`Dict[str, str]`) – User information dict (must have login and privilege keys)
- `privilege` (`Optional[str]`) – Override privilege (optional, uses user_data[privilege] by default)

Return type

`str`

Returns
JWT token string

Example

```
>>> token = create_token({"login": "admin", "privilege": "admin"})
>>> print(token[:20])
eyJhbGciOiJIUzI1NiIs...
```

`webserver.get_logger`(*name*)

Get a custom logger that writes to error.log, for specialized logging needs.

Parameters
name (*str*) – Logger name.

Returns
Logger instance for custom usage.

Return type
logging.Logger

`webserver.require_auth`(*required_privilege=None*)

FastAPI dependency factory: Require authentication with optional privilege check.

Parameters
required_privilege (`Optional[str]`) – Minimum required privilege (PRIVILEGE_ADMIN or PRIVILEGE_ENDPOINT) If None, any authenticated user is allowed

Returns
FastAPI dependency function

Raises
HTTPException – If unauthorized (401) or insufficient privilege (403)

Usage:
# Any authenticated user @app.get(/users, dependencies=[Depends(require_auth())])

# Admin only @app.delete(/users/{user_id}, dependencies=[Depends(require_auth(PRIVILEGE_ADMIN))])

`webserver.verify_token`(*token*)

Verify and decode a JWT token.

Parameters
token (`str`) – JWT token string

Return type
`Dict[str, Any]`

Returns
Decoded payload dict with login and privilege

Raises
HTTPException – If token is invalid or expired (401)

Example

```
>>> payload = verify_token("eyJhbGci...")
>>> print(payload['login'])
admin
```

PYTHON MODULE INDEX