Auth Service Documentation

Release v1.0.0

Дымников Михаил (dym-dino)

Mar 31, 2025

CONTENTS

CHAPTER

ONE

APP

ICCEM backend API

1.1 logger

1.1.1 logging

1.2 models

Models Module

Defines data models used across the application.

1.2.1 result

Uniform Response

Defines a Pydantic model for uniform requests response.

class app.models.result.UniformResponse(* (Keyword-only parameters separator (PEP 3102)), status_code: int = 200, result: str, data: Any = None)

Bases: BaseModel

Uniform API response structure.

status_code
HTTP status code of the response (default is 200).

Type
int

result
A message describing the outcome of the operation.

Type
str

data
Additional data returned by the API.

Type
Any

status_code: int

result: str

data: Any

1.2.2 token

Token Request

Defines a Pydantic model for token authentication requests.

class app.models.token.RefreshRequest(*, refresh_token: str)

Bases: BaseModel

refresh_token: str

class app.models.token.TokenRequest(*, secret_hash: str, role: str, login: str | None = None, password: str | None = None)

Bases: BaseModel

Model representing a token request for authentication.

secret_hash

A secret hash used for authentication.

Type

str

role

The user's role, e.g., "admin" or "operator".

Type

str

login

The login identifier for the user, if applicable.

Type

Optional[str]

password

The user's password, if applicable.

Type

Optional[str]

secret_hash: str

role: str

login: str | None

password: str | None

1.2.3 user

Users Models

Defines SQLAlchemy and Pydantic models for users entities.

class app.models.user.User(**kwargs)

Bases: Base

id

login

password

role

permissions

class app.models.user.UserCreate(*, login: str, password: str, role: str, permissions: List[str] = None)

Bases: UsersBase

class app.models.user.UserInDB(*, login: str, password: str, role: str, permissions: List[str] = None, id: str | None = None)

Bases: UsersBase

id: str | None

class app.models.user.UsersBase(*, login: str, password: str, role: str, permissions: List[str] = None)

Bases: BaseModel

login: str

password: str

role: str

permissions: List[str]

1.3 routers

Routers Package

Contains API route definitions for various modules.

1.3.1 admin

users

1.3.2 public

ping

1.3.3 auth

1.4 services

Services Package

Provides auxiliary services for the application.

1.4.1 backup

1.5 tests

1.5.1 conftest

1.5.2 test_auth

Tests for token generation and refresh endpoints.

app.tests.test_auth.test_get_token_user(client: TestClient)

Test generating access and refresh tokens for a user.

- Sends a POST request to /token with role 'user', login, password, and valid secret_hash.
- Expects a 200 OK response.
- Asserts that both access_token and refresh_token are present in the response.

app.tests.test_auth.test_get_token_admin_create_and_login(client: TestClient)

Test token generation for an admin with user auto-creation and authentication.

- Sends a POST request with admin role, login, password, and secret_hash.
- Expects user creation on first call and authentication on the second.
- Verifies both calls return valid tokens.

app.tests.test_auth.test_invalid_secret_key(client: TestClient)

Test access denial when an invalid secret_hash is used.

- Sends a POST request to /token with an incorrect secret_hash.
- Expects 403 Forbidden response.
- Verifies the error message mentions invalid token secret.

app.tests.test_auth.test_refresh_token(client: TestClient)

Test refreshing an access token using a valid refresh token.

- Sends a POST request to /token to obtain tokens.
- Uses the refresh_token in a POST request to /refresh.
- Expects 200 OK and a new access_token in the response.

app.tests.test_auth.test_invalid_refresh_token_type(client: TestClient, user_token: str)

Test error when using an access token instead of a refresh token for refreshing.

- Sends an access_token in a POST request to /refresh.
- Expects 403 Forbidden response.
- Checks for error message "Expected refresh token".

1.5.3 test_public

Tests for the /me public endpoint.

app.tests.test_public.test_get_current_user_payload(client: TestClient, user_token: str)

Test successful retrieval of current user payload using valid JWT.

- Sends a GET request to /me with a valid Authorization header.
- Expects a 200 OK response.

- Asserts that the returned payload contains the correct role.

app.tests.test_public.test_get_current_user_no_token(client: TestClient)

Test error when accessing /me endpoint without Authorization header.

- Sends a GET request to /me without any token.
- Expects 401 Unauthorized response.
- Verifies that the response includes 'Missing Authorization header'.

1.5.4 test_users

Tests for user creation, deletion, listing, and access control.

app.tests.test_users.test_create_user(client: TestClient, admin_token: str)

Test user creation via /create_user endpoint.

- Sends a POST request with login, password, role, and permissions.
- Expects a 200 OK response and correct user data in response body.

app.tests.test_users.test_create_user_duplicate(client: TestClient, admin_token: str)

Test duplicate user creation.

- Creates a user with a specific login.
- Attempts to create the same user again.
- Expects a 400 Bad Request with a message about duplicate login.

app.tests.test_users.test_get_users_list(client: TestClient, admin_token: str)

Test retrieving a paginated list of users.

- Creates 15 users.
- Sends a GET request with limit and offset parameters.
- Expects a list of users with length <= limit.

app.tests.test_users.test_remove_user(client: TestClient, admin_token: str)

Test user deletion by ID.

- Creates a user.
- Deletes the user using their ID.
- Expects a 200 OK response with confirmation message.

app.tests.test_users.test_remove_nonexistent_user(client: TestClient, admin_token: str)

Test deletion of a non-existent user.

- Sends a DELETE request with a random UUID.
- Expects a 404 Not Found response.

app.tests.test_users.test_update_user_permissions(client: TestClient, admin_token: str)

Test updating user permissions.

- Creates a user with initial permissions.
- Sends a PUT request to update their permissions.
- Expects a 200 OK response and updated permission list in the result.

1.6 utils

Utils Module

Provides utility functions and helpers used across the application

1.6.1 auth

Auth Module

Provides utility functions and helpers used across the application

hash

Hash Utility

Provides a function for generating a SHA-256 hash from a given string.

app.utils.auth.hash.hash_str(s: str) → str

Generates a SHA-256 hash for the provided string.

Parameters
s (str) – The input string to hash.

Returns
The hexadecimal SHA-256 hash of the input string.

Return type
str

jwt_handler

JWT Handler

Provides functions for creating and verifying JWT tokens for authentication, role validation, and permission-based access control.

app.utils.auth.jwt_handler.get_token(authorization: str = Depends(APIKeyHeader)) → str

Extracts token from Authorization header.

app.utils.auth.jwt_handler.create_access_token(data: Dict[str, Any], expires_delta: timedelta = datetime.timedelta(seconds=900)) → str

Creates a JWT access token.

app.utils.auth.jwt_handler.create_refresh_token(data: Dict[str, Any], expires_delta: timedelta = datetime.timedelta(days=7)) → str

Creates a JWT refresh token.

app.utils.auth.jwt_handler.verify_token(token: str = Depends(get_token), expected_type: str = 'access') → Dict[str, Any]

Verifies a JWT token (access or refresh) and returns its payload.

app.utils.auth.jwt_handler.verify_refresh_token(token: str = Depends(get_token)) → Dict[str, Any]

Verifies a JWT refresh token and returns its payload.

app.utils.auth.jwt_handler.require_permission(permission: str)

Dependency that checks if the user has a specific permission in token payload. Usage:

@router.get("/secure", dependencies=[Depends(require_permission("read_users"))])

static_protection

Static protections

Secured code docs route with login and password

class app.utils.auth.static_protection.ProtectedStaticFiles(*, directory: str | PathLike[str] | None = None, packages: list[str | tuple[str, str]] | None = None, html: bool = False, check_dir: bool = True, follow_symlink: bool = False)

Bases: StaticFiles

async get_response(path: str, scope)

Returns an HTTP response, given the incoming path, method and request headers.

swagger_auth

Swagger Auth

Module for protecting access to Swagger documentation using HTTP Basic authentication.

app.utils.auth.swagger_auth.get_swagger_basic_auth(credentials: HTTPBasicCredentials = Depends(HTTPBasic)) → str

Dependency for HTTP Basic authentication to access Swagger documentation.

Parameters
credentials (HTTPBasicCredentials) – User-provided credentials.

Returns
The authenticated username.

Return type
str

Raises
HTTPException – If the provided credentials are invalid.

1.6.2 base_handler

1.6.3 rate_limit

Rate Limiting

Provides global rate limiting configuration using slowapi.

1.7 config

Config Settings

Module for loading application configuration settings from a .env file.

class app.config.Settings

Bases: object

SECRET_KEY: str = 'MegaCyberDragon42_HacksTheNeonMatrixWhileRidingLaserSharks!'

ADMIN_SWAGGER_PASSWORD: str = 'HocrRMMR8PJv8kahGbrXS-J'

ADMIN_SWAGGER_LOGIN: str = 'iccem_admin'

POSTGRES_DB: str = 'iccem_logs'

POSTGRES_USER: str = 'iccem_admin'

POSTGRES_PASSWORD: str = 'xVhdRFEOHLyWavEc_C2bnuE'

POSTGRES_PORT: str = '5432'

POSTGRES_HOST: str = 'postgres'

POSTGRES_URL: str =
'postgresql://iccem_admin:xVhdRFEOHLyWavEc_C2bnuE@postgres:5432/iccem_logs'

BACKUP_PATH: str = '/home/iccem/backups'

DOCS_DIR: Path =
WindowsPath('C:/Users/miald/storage/Progarmming/dino-projects/AuthService/docs/html')

LOG_TO_DB: bool = True

TESTING: bool = False

1.8 database

PostgreSQL Connection

Connects to PostgreSQL database using SQLAlchemy.

app.database.get_db()

1.9 main