# Python Code Documentation

## Overview

This document provides documentation for the Python code used in a FastAPI application for managing todos.

### Authentication

#### Register User

* **Endpoint:** /register
* **Method:** POST
* **Description:** Registers a new user with a username and password.
* **Request Body:**
  + username: String (required) - The username of the user.
  + password: String (required) - The password of the user.
* **Response:**
  + Status: 200 OK
  + Content: { "message": "User registered successfully" }

#### Obtain Access Token

* **Endpoint:** /token
* **Method:** POST
* **Description:** Logs in a user and returns an access token.
* **Request Body:**
  + username: String (required) - The username of the user.
  + password: String (required) - The password of the user.
* **Response:**
  + Status: 200 OK
  + Content: { "access\_token": "JWT\_TOKEN", "token\_type": "bearer" }

### To-do Operations

#### Create To-do

* **Endpoint:** /todos/
* **Method:** POST
* **Description:** Creates a new to-do item.
* **Request Body:**
  + title: String (required) - The title of the to-do.
  + description: String (required) - The description of the to-do.
  + done: Boolean (optional) - The status of completion (default is False).
* **Response:**
  + Status: 200 OK
  + Content: { "message": "Todo created successfully", "todo\_id": "TODO\_ID" }

#### Read To-dos

* **Endpoint:** /todos/
* **Method:** GET
* **Description:** Retrieves all to-do items.
* **Response:**
  + Status: 200 OK
  + Content: List of to-do items.

#### Read To-do by Title

* **Endpoint:** /todos/{todo\_title}
* **Method:** GET
* **Description:** Retrieves a specific to-do item by its title.
* **Response:**
  + Status: 200 OK
  + Content: The to-do item.

#### Update To-do

* **Endpoint:** /todos/{todo\_title}
* **Method:** PUT
* **Description:** Updates a specific to-do item.
* **Request Body:**
  + title: String (optional) - The new title of the to-do.
  + description: String (optional) - The new description of the to-do.
  + done: Boolean (optional) - The new status of completion.
* **Response:**
  + Status: 200 OK
  + Content: { "message": "Todo updated successfully" }

#### Delete To-do

* **Endpoint:** /todos/{todo\_title}
* **Method:** DELETE
* **Description:** Deletes a specific to-do item by its title.
* **Response:**
  + Status: 200 OK
  + Content: { "message": "Todo deleted successfully" }

### Authentication Dependency

To access the above endpoints, the user must include an access token in the request header. The access token can be obtained by logging in via the /token endpoint.

### Token Verification

The access token must be included in the Authorization header of the request in the format Bearer JWT\_TOKEN.

### Requirements

1. **FastAPI**: The main framework used for building the API.
2. **PyJWT**: For encoding and decoding JSON Web Tokens (JWT) for authentication.
3. **Passlib**: For secure password hashing.
4. **Pymongo**: A MongoDB driver for Python, used for interacting with the MongoDB database.
5. **OAuth2PasswordBearer**: A FastAPI security dependency for handling OAuth2 password flow for token-based authentication.
6. **BaseModel**: From the Pydantic library, used for defining data models with validation and serialization capabilities.
7. **OAuth2PasswordRequestForm**: A FastAPI class for handling OAuth2 password request form data.
8. **Depends**: A FastAPI class used for dependency injection.
9. **status**: A module from FastAPI for defining HTTP status codes.
10. **timedelta**: From the datetime module, used for calculating token expiration time.

### How To Run

1. Download All libraries from above requirements
2. Run the FastAPI application using the following command: uvicorn main:app --reload.
3. Create MongoDb Dabase URI to connect database.

## Variables

### FastAPI App

* **Description**: The main instance of the FastAPI application.
* **Type**: FastAPI object.

### MongoDB Client

* **Description**: The client object for connecting to the MongoDB database.
* **Type**: MongoClient object.

### MongoDB Database

* **Description**: The database object representing the MongoDB database.
* **Type**: Database object.

### MongoDB Collection

* **Description**: The collection object representing the MongoDB collection for storing todos.
* **Type**: Collection object.

### Secret Key for Token

* **Description**: Secret key used for encoding and decoding JWT tokens.
* **Type**: str.

### Algorithm for Token

* **Description**: The algorithm used for encoding and decoding JWT tokens.
* **Type**: str.

### Access Token Expiry

* **Description**: Time duration for which access tokens are valid (in minutes).
* **Type**: int.

### OAuth2 Scheme

* **Description**: OAuth2 password bearer scheme for token authentication.
* **Type**: OAuth2PasswordBearer object.

### Password Context

* **Description**: Context object for hashing and verifying passwords.
* **Type**: CryptContext object.

### User Model for Registration

* **Description**: Pydantic model representing user data for registration.
* **Type**: BaseModel subclass.

### User Model for Authentication

* **Description**: Pydantic model representing user data retrieved from the database for authentication.
* **Type**: BaseModel subclass.

### Todo Model

* **Description**: Pydantic model representing todo data.
* **Type**: BaseModel subclass.

### OAuth2 Password Request Form

* **Description**: Pydantic model representing the form data required for OAuth2 password authentication.
* **Type**: OAuth2PasswordRequestForm class.

## Routes

### User Registration Route (“/register”)

* **Description**: Route for user registration.
* **Type**: POST endpoint.

### Token Generation Route (“/token”)

* **Description**: Route for generating access tokens.
* **Type**: POST endpoint.

### Read Todos Route (“/todos/”)

* **Description**: Route for reading all todos.
* **Type**: GET endpoint.

### Create Todo Route (“/todos/”)

* **Description**: Route for creating a new todo.
* **Type**: POST endpoint.

### Read Todo Route (“/todos/{todo\_title}”)

* **Description**: Route for reading a specific todo by its title.
* **Type**: GET endpoint.

### Update Todo Route (“/todos/{todo\_title}”)

* **Description**: Route for updating a specific todo by its title.
* **Type**: PUT endpoint.

### Delete Todo Route (“/todos/{todo\_title}”)

* **Description**: Route for deleting a specific todo by its title.
* **Type**: DELETE endpoint.