To structure your AI-powered voice assistant code, it's essential to break down the project into distinct modules to ensure clarity, scalability, and ease of maintenance. Below is a step-by-step plan on how to structure your code:

1. Project Directory Structure

Here's a suggested directory structure for your voice assistant:

```
voice-assistant/
│
├── config/
│   ├── settings.py          # Configuration for API keys, paths,
etc.
│   └── constants.py         # Define constants like admin password,
available apps, etc.
│
├── core/
│   ├── __init__.py
│   ├── assistant.py         # Main class that controls the
assistant logic
│   ├── command_processor.py # Processes commands, decides actions
│   └── authentication.py    # Handles user authentication
│
├── speech/
│   ├── __init__.py
│   ├── speech_recognition.py # Handles speech-to-text (using
libraries like SpeechRecognition or Whisper)
│   ├── text_to_speech.py    # Converts text to speech (using
pyttsx3 or Coqui TTS)
│   └── wakeword.py          # Handles wakeword detection (e.g.,
Porcupine or Snowboy)
│
├── commands/
│   ├── __init__.py
│   ├── web_search.py        # For performing web searches
│   ├── app_control.py       # For opening applications
│   ├── file_management.py   # For file creation and management
│   └── system_control.py    # For advanced system functions (volume
control, etc.)
│
├── logs/
```

```
│   ├── __init__.py
│   └── logger.py            # Handles logging of assistant's
actions
│
└── main.py                  # Entry point for starting the
assistant (run this file)
```

2. Code Breakdown by Module

config/settings.py

- **Purpose**: Central configuration file for constants, API keys, admin passwords, etc.
- **What to include**:
  - API keys for services like Google Custom Search.
  - Admin password and other security credentials.
  - Default voice settings for text-to-speech.

```
# settings.py
ADMIN_PASSWORD = "your_admin_password_here"
WAKEWORD = "Hey Assistant"
VOICE_ENGINE = "pyttsx3"
```

core/assistant.py

- **Purpose**: This is the main class where you control the flow of the assistant.
- **What to include**:
  - Initialize all necessary modules (speech recognition, TTS, command processor, etc.).
  - Manage the wake word detection and trigger appropriate actions.
  - Handle both **Admin Mode** and **Basic Mode** based on user authentication.

```
# assistant.py
from speech.speech_recognition import listen
from commands.command_processor import process_command
from core.authentication import authenticate_user
from speech.text_to_speech import speak

class VoiceAssistant:
```

```
    def __init__(self):
        self.is_authenticated = False
        self.is_admin = False

    def start(self):
        while True:
            print("Listening for wakeword...")
            command = listen()  # Listen for command
            if command.lower() == "hey assistant":
                self.authenticate()
                self.listen_for_commands()

    def authenticate(self):
        password = authenticate_user()
        if password == settings.ADMIN_PASSWORD:
            self.is_authenticated = True
            self.is_admin = True
            speak("Admin mode activated.")
        else:
            self.is_authenticated = True
            self.is_admin = False
            speak("Basic mode activated.")

    def listen_for_commands(self):
        command = listen()  # Listen for actual command
        process_command(command, self.is_admin)
```

speech/speech_recognition.py

- **Purpose**: Handle all aspects of speech recognition (convert speech to text).
- **What to include**:
    - Use SpeechRecognition or **Whisper** for speech-to-text.
    - Integration with the wakeword detector.

```
# speech_recognition.py
import speech_recognition as sr

def listen():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
```

```
        audio = recognizer.listen(source)
        try:
            command = recognizer.recognize_google(audio)
            return command
        except Exception as e:
            print("Could not recognize speech:", e)
            return ""
```

speech/text_to_speech.py

- **Purpose**: Convert text into speech for the assistant to respond.
- **What to include**:
    - Use **pyttsx3** or **Coqui TTS** for generating speech responses.

```
# text_to_speech.py
import pyttsx3

def speak(text):
    engine = pyttsx3.init()
    engine.say(text)
    engine.runAndWait()
```

speech/wakeword.py

- **Purpose**: Detect the wake word, like "Hey Assistant".
- **What to include**:
    - Integrate **Porcupine** or **Snowboy** for efficient wakeword detection.

```
# wakeword.py
import pvporcupine
import pyaudio

def detect_wakeword():
    porcupine = pvporcupine.create(keywords=["Hey Assistant"])
    audio = pyaudio.PyAudio()
    stream = audio.open(rate=porcupine.sample_rate, channels=1,
format=pyaudio.paInt16, input=True)

    while True:
        pcm = stream.read(porcupine.frame_length)
```

```
        result = porcupine.process(pcm)
        if result >= 0:
            return "Hey Assistant"
```

core/authentication.py

- **Purpose**: Handle user authentication (Admin vs Basic mode).
- **What to include**:
    - Prompt for the admin password, and determine access level.

```
# authentication.py
from getpass import getpass

def authenticate_user():
    password = getpass("Enter your password: ")
    return password
```

commands/command_processor.py

- **Purpose**: Process the user command and take the appropriate action.
- **What to include**:
    - Based on the mode (Admin or Basic), trigger corresponding functions.

```
# command_processor.py
from commands.web_search import web_search
from commands.app_control import open_application
from commands.file_management import create_file
from commands.system_control import control_volume

def process_command(command, is_admin):
    if "open" in command:
        open_application(command)
    elif "search" in command:
        web_search(command)
    elif "create" in command:
        create_file(command)
    if is_admin:
        if "volume" in command:
            control_volume(command)
```

commands/web_search.py

- **Purpose**: Perform a web search.
- **What to include**:
    - Use **Google Custom Search API** or a basic web scraping library.

```
# web_search.py
import webbrowser

def web_search(query):
    search_query = query.replace("search", "")
    url = f"https://www.google.com/search?q={search_query}"
    webbrowser.open(url)
```

logs/logger.py

- **Purpose**: Handle logging of actions and commands.
- **What to include**:
    - Log every action taken by the assistant (especially admin commands).

```
# logger.py
import logging

def setup_logger():
    logging.basicConfig(filename="assistant_log.txt",
level=logging.INFO)

def log_action(action):
    logging.info(action)
```

3. Main Execution (main.py)

This is where everything comes together. You'll initialize the assistant, start listening for the wake word, and authenticate the user.

```
# main.py
from core.assistant import VoiceAssistant

if __name__ == "__main__":
    assistant = VoiceAssistant()
```

```
assistant.start()
```

4. Final Thoughts

- **Separation of Concerns**: Each module has a clear responsibility (speech, commands, authentication, etc.), making it easy to maintain and extend.
- **Scalability**: As you add more commands or functionalities, you can easily extend the `commands` directory.
- **Flexibility**: This structure supports AI-based upgrades, like adding more intelligent command processing or using more advanced models for speech recognition and response.

With this structure, you should be able to modularly add new features or improve existing ones. Let me know if you need more specific code examples or further clarification!