Projet de fin de module :

LANGAGE PYTHON

CLASSE : DATA INE 1

Développement d'un Bot Automatisé pour Discord

Réalisé par :

AIT-ABDELOUAHAB MEHDI & ASSADDIKI Mohamed Khalil

Encadré par :

Pr BAINA Amine

1. Introduction et Contexte

1.1. Présentation synthétique du projet

Dans un contexte où les communautés en ligne ne cessent de croître, notre projet consiste à développer un bot Python pour Discord. Ce bot automatise la modération de communautés, tout en offrant des fonctionnalités gaming innovantes comme le web scraping pour les meilleures promotions et des recommandations personnalisées basées sur l'intelligence artificielle.

1.2. Objectifs principaux

- Automatiser la modération (ban, mute, anti-spam, anti-gros mots, suppression massive).
- Fournir des informations utiles pour les gamers et les passionnés de technologie via des notifications quotidiennes ou à la demande
- Promouvoir les bonnes pratiques de développement en Python, notamment dans l'interaction avec les API, le web scraping et l'automatisation des taches, utilisation de ".env" pour la gestion des cles privees.

1.3. Contexte

Les bots jouent un rôle crucial sur des plateformes comme Discord, améliorant l'expérience utilisateur tout en réduisant la charge des administrateurs. Discord, avec ses millions d'utilisateurs actifs, est une plateforme incontournable dans le monde du gaming et des communautés en ligne. Ce projet vise à optimiser la gestion communautaire et à enrichir l'expérience utilisateur grâce à des services pratiques et innovants.

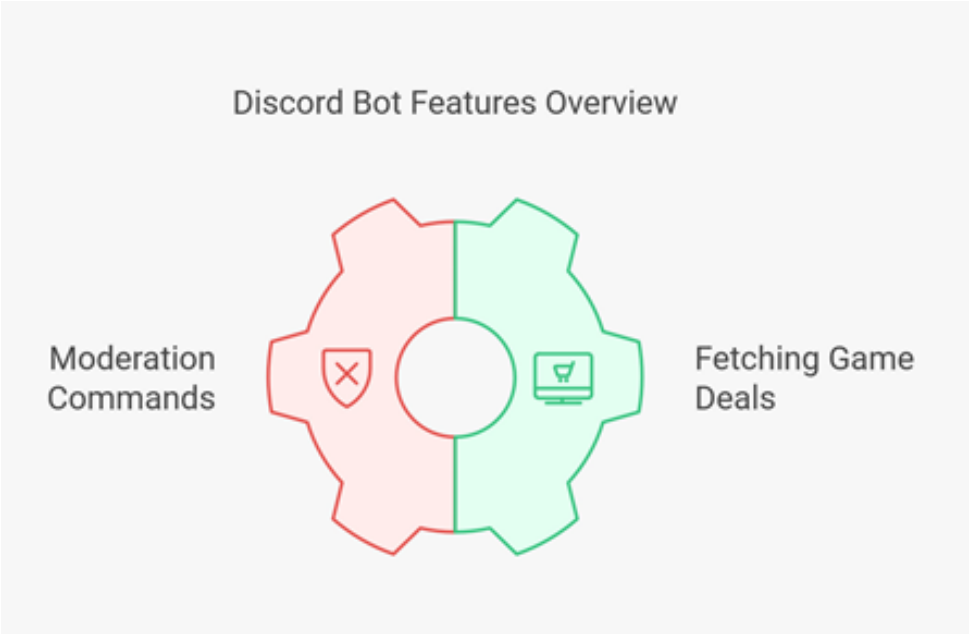

Figure 1: Fonctionnalité principaux.

2. Conception et Fonctionnalités

2.1. Choix de la Plateforme

Discord a été choisi pour son adoption massive dans la sphère gaming et tech. Le bot cible les administrateurs et les membres des serveurs souhaitant une gestion simplifiée et un accès rapide à des informations pertinentes comme les promotions et les actualités technologiques.

2.2. Description des Fonctionnalités

Modération:

- **Anti-spam**: Surveillance et suppression automatique des messages suspects.
- **Ban/Mute**: Actions automatisées en cas de comportements inappropriés.
- **Filtrage de mots interdits**: Blocage des propos offensants.
- **Suppression massive (Purge)**: Nettoyage rapide des canaux

Gaming et Promotions:

- **Promotions gaming**: Récupération des meilleures offres promotionnelles avec le web scraping pour des notifications quotidiennes ou à la demande.
- **Recommandations technologiques**: Suggestions personnalisées d'outils gaming et tech, basées sur l'IA.

2.3. Architecture

Le bot repose sur une architecture modulaire intégrant:

- **API Discord**: Interaction avec les serveurs et les utilisateurs.
- **Bibliothèques Python**: Modules tels que ‘discord.py‘ pour l’intégration
- **BeautifulSoup et Selenium**: le web scraping
- **Requests**: l’interaction avec des API externes comme GEMINI.
- **Flux de données**: Gestion des entrées utilisateur, traitement des requêtes et envoi de réponses ou notifications.

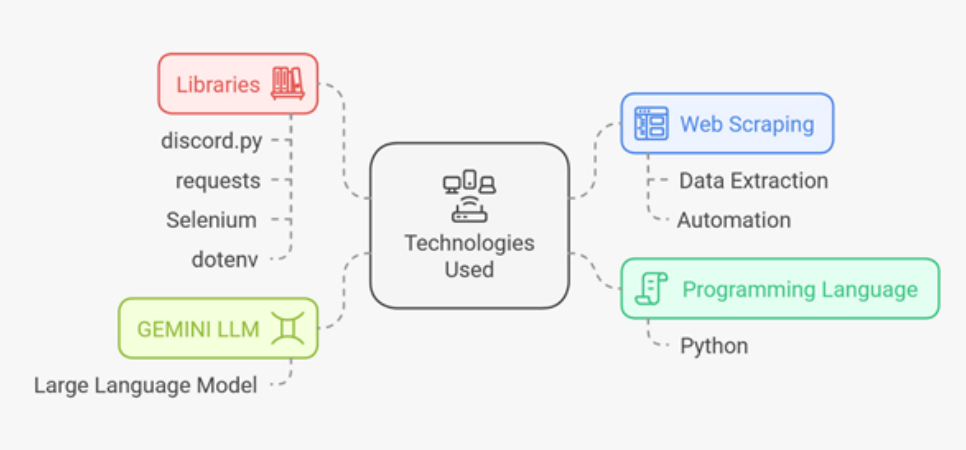

Figure 2: Téchnologies utilisées

3. Implémentation Technique

3.1. Technologies et Bibliothèques

- **discord.py** : Gestion de l’API Discord.
- **BeautifulSoup** : Extraction des données web.
- **Selenium** : Automatisation de tâches complexes de navigation web, permettant de gérer des sites dynamiques comme des formulaires ou des contenus chargés en JavaScript.
- **Requests** : Interaction avec des API externes pour envoyer et recevoir des données, facilitant ainsi la communication avec des services IA comme GEMINI.

3.2. Développement

Une approche par fonctionnalité a été adoptée, appuyée par l’utilisation de Git pour une collaboration efficace en équipe. Cette méthode a permis une gestion structurée du projet, incluant le suivi des versions, la répartition des tâches et la résolution rapide des conflits de code.

3.3. Extraits du Code

```python
# function that scrapes the data from the website
#@return format: [{name,deal_value,old_value, image, link}...]
Codeium: Refactor | Explain | Generate Docstring | ×
def scrape_deals(max_deals=10):
    driver = webdriver.Chrome()
    websitelink = "https://gg.deals"
    driver.get(f"{websitelink}/deals/") # navigate to the website
    WebDriverWait(driver,5).until(EC.presence_of_element_located((By.CLASS_NAME, "deal-list-item")))
    elements = BeautifulSoup(driver.page_source, "html.parser").find_all("div", class_="deal-list-item", limit=max_deals)
    driver.quit()
    deals = []
    for element in elements :
        # scrapping the image
        try :
            image = element.select_one(".game-image .main-image picture img")
            image = image['srcset'].split(',')[1].replace(" 2x", "") or image['src']
            #Deal Value and name
            deal_value = element.select_one(".price-wrapper .game-price-new").text
            old_value = element.select_one("span.price-label.price-old")
            if old_value is not None:
                old_value = old_value.text
            else:
                old_value = "Unknown"
            game_name = element.select_one(".game-info-wrapper .game-info-title-wrapper .title").text
            #deal link
            deal_link = websitelink + element.select_one(".game-cta .shop-link")['href']
            deals.append({
                "name": game_name,
                "deal_value": deal_value,
                "old_value": old_value,
                "image": image,
                "link": deal_link
            })
        except Exception as e:
            print("Error during scrapping deals: ", e)
            continue

    return deals
```

Figure 3: Web Scraping

```python
def clean_api_data(data):
    try :
        cleaned_data = data['candidates'][0]['content']['parts'][0]['text'].replace("```json","").replace("```","")
        start_index = cleaned_data.find('[')
        end_index = cleaned_data.find(']')
        if start_index != -1 and end_index != -1:
            cleaned_data = json.loads(cleaned_data[start_index:end_index+1])
        return cleaned_data
    except Exception as e:
        print("Error during cleaning returned Data: ", e)
        return None

def query_gemini_api(prompt):
    payload = {'contents': [{'parts': [{'text': prompt}]}]}
    try:
        response = requests.post(GEMINI_PROMPT_URL, headers={"Content-Type": "application/json"}, json=payload)
        response.raise_for_status()
        data =clean_api_data(response.json())
        return data
    except requests.exceptions.RequestException as e:
        print("A request error has occurred:", e)
        return None
    except Exception as e:
        print("An unexpected error occurred:", e)
        return None

# fully functional, looks for recommendations through gemini api
def get_recommendations(product, budget):
    # sanitizing the prompt
    sanitize_request = f"answer with only one of the following options (No | hardware | Software), if {product} is related to electronics,technology or gaming ?"
    type = query_gemini_api(sanitize_request).strip().lower().replace(".","")
    if type == "no":
        return [], "Not allowed"
    elif type == "software" or type == "hardware":
        # creating the prompt
        prompt = f"""Provide a JSON array of {product} recommendations (up to 5) within a ${budget} budget.
        The JSON format should contain the fields: "name" (with brand), "actual_price", and "desc". No extra text."""
        recommendations = query_gemini_api(prompt)
        return recommendations, type
    else :
        raise Exception("Unexpected response from API")
```

Figure 4: Code de recommendation

```python
async def anti_spam(message):
    """
    Anti-spam function to detect and mute spamming users.
    """
    if message.author.bot:
        return

    user_id = str(message.author.id)  # Convert user ID to string for JSON compatibility
    current_time = time.time()

    # Load user message timestamps from the file
    try:
        with open(user_messages_path, "r") as f:
            user_messages = json.load(f)
            # Convert lists to deque for efficient processing
            user_messages = {k: deque(v) for k, v in user_messages.items()}
    except (FileNotFoundError, json.JSONDecodeError):
        user_messages = {}

    # Initialize deque for the user if not present
    if user_id not in user_messages:
        user_messages[user_id] = deque()

    # Add the current message timestamp
    user_messages[user_id].append(current_time)

    # Remove timestamps older than the time window
    while user_messages[user_id] and current_time - user_messages[user_id][0] > TIME_WINDOW:
        user_messages[user_id].popleft()
```

Figure 5: Code de modération

```python
# Save the updated user messages back to the file
try:
    with open(user_messages_path, "w") as f:
        # Convert deques to lists for JSON serialization
        json.dump({k: list(v) for k, v in user_messages.items()}, f, indent=4)
except Exception as e:
    print(f"Error saving user messages: {e}")
    return

# Check if the user exceeds the spam threshold
if len(user_messages[user_id]) > SPAM_THRESHOLD:
    await message.channel.send(f"🚨 {message.author.mention}, you are spamming! Please slow down.")
    await message.delete()

    # Mute the user
    await mute(
        message,
        member=message.author,
        duration=MUTE_DURATION,
        reason="Spamming messages"
    )
    await message.channel.send(
        f"🔇 {message.author.mention} has been muted for {MUTE_DURATION // 60} minutes due to spamming."
    )
```

Figure 6: Code de modération

3.4. Tests et Validation

- **Scénarios réels** : Simulation d'utilisation dans un serveur Discord test.
- **Résultats** : Bonne performance et précision des réponses, avec un temps de réponse moyen inférieur à 500ms.

4. Conclusion et Perspectives

4.1. Résultats

Le bot offre une modération efficace et des services de grande valeur pour les utilisateurs. Il répond aux besoins des administrateurs tout en enrichissant l'expérience communautaire.

4.2. Leçons Apprises

- **Défis** : Intégration des modules IA et optimisation des performances.
- **Compétences acquises** : Maîtrise avancée des bibliothèques Python et gestion des flux de données.

4.3. Perspectives d'Amélioration

- Extension des fonctionnalités IA pour des recommandations plus précises.
- Mise en place de liens affiliés pour générer des revenus supplémentaires.
- Ajout d'un flux d'actualités sur les jeux vidéo et la technologie pour maintenir l'intérêt des utilisateurs.
- Compatibilité avec d'autres plateformes comme Slack ou Microsoft Teams.

5. Annexes

- Documentation de la bibliothèaue *Discord.py* : https://discordpy.readthedocs.io/en/stable/
- Tutoriels sur le web scraping et l'IA.
- GEMINI API : https://ai.google.dev