//THIS DOCUMENT CONTAINS THE PROJECT REPORT AND IS DIVIDED IN SECTION 1 AND 2 AS PER ASSIGNMENT//

• SECTION 1:

In this section we discuss how to implement the functionality which is mentioned optional as far as coding is concerned

1) To implement email verification upon registration you can use the Flask-Mail extension along with a token-based verification system.

Update Your Flask App Configuration
In your Flask app (app.py), update the configuration to include mail settings:

~~~
app.config['MAIL_SERVER'] = 'smtp.example.com'  # Update with your SMTP server
app.config['MAIL_PORT'] = 587  # Update with your SMTP port
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USE_SSL'] = False
app.config['MAIL_USERNAME'] = 'your_email@example.com'  # Update with your email
app.config['MAIL_PASSWORD'] = 'your_email_password'  # Update with your email password
app.config['MAIL_DEFAULT_SENDER'] = 'your_email@example.com'

mail = Mail(app)
```
~~~

In your user registration route (`/register`), generate a verification token and send an email with a verification link:

~~~
```python
from itsdangerous import URLSafeTimedSerializer  # Install it using: pip install itsdangerous
# Update the route
@app.route("/register", methods=["POST"])
def register_user():
    email = request.json["email"]
~~~

```
    password = request.json["password"]

    user_exists = User.query.filter_by(email=email).first()

    if user_exists:
        return jsonify({"error": "Email already exists"}), 409

    # Generate a unique verification token
    serializer = URLSafeTimedSerializer(app.config['SECRET_KEY'])
    verification_token = serializer.dumps(email, salt='email-verification')

    # Send the verification email
    verification_link = url_for('verify_email', token=verification_token, _external=True)
    subject = 'Confirm Your Email'
    body = f'Click the following link to verify your email: {verification_link}'

    message = Message(subject, recipients=[email], body=body)
    mail.send(message)

    hashed_password = bcrypt.generate_password_hash(password)
        new_user = User(email=email, password=hashed_password, about="SAMPLE
ABOUT ME")
    db.session.add(new_user)
    db.session.commit()

    return jsonify({
        "message": "Registration successful. Check your email for verification."
    })

# Add a new route for email verification
@app.route("/verify-email/<token>")
def verify_email(token):
    serializer = URLSafeTimedSerializer(app.config['SECRET_KEY'])

    try:
        email = serializer.loads(token, salt='email-verification', max_age=3600)  # 1 hour
expiration
        # Update user's email_verified field in the database
        user = User.query.filter_by(email=email).first()
        user.email_verified = True
        db.session.commit()

        return "Email verified successfully. You can now log in."
    except Exception as e:
        print(e)
        return "Email verification failed."

# ...
```
Create a New Route for Email Verification
Create a new route (`/verify-email/<token>`) to handle email verification. When a user clicks the verification link, this route will update the user's `email_verified` field in the database.

2)Implementing a password reset functionality involves similar considerations as email verification.

1. Flask-Mail for Sending Reset Password Emails:
   - Pros:
     - Utilizes Flask integration for sending emails.
     - Straightforward setup for sending reset instructions.
   - Cons:
     - Limited compared to specialized email services.

2. Secure Token Generation for Password Reset:
   - Pros:
     - URLSafeTimedSerializer provides secure token generation with an expiration time.
     - Tokens can be tied to a specific user and purpose (password reset).
   - Cons:
     - Basic token functionality.

# Alternatives:

1. Email Sending Alternatives:
   - Consider other Flask extensions or services for sending emails:
     - [Flask-Sendmail](https://pythonhosted.org/Flask-Sendmail/)

- [smtplib (Python Standard Library)](https://docs.python.org/3/library/smtplib.html)
    - Third-party services like SendGrid for improved deliverability.
2. Token Generation Alternatives:
   - Explore other token generation libraries:
[itsdangerous.TimestampSigner](https://itsdangerous.palletsprojects.com/en/2.1.x/api/#itsdangerous.TimestampSigner)
    - [PyJWT (JSON Web Tokens)](https://pyjwt.readthedocs.io/en/stable/)

# Overall Considerations:
1. Security:
   - Ensure HTTPS is used for communication.
   - Implement proper token handling and storage.
   - Use a secure random key for token signing.

2. Token Expiration:
   - Set an appropriate expiration time for security. A shorter expiration time enhances security but might inconvenience users.

3. User Experience:
   - Provide clear instructions in the password reset email.
   - Design the password reset page to be user-friendly.

4. Logging and Monitoring:
   - Implement logging and monitoring to track email delivery and user interactions with password reset links.

5. Email Templates:
   - Customize your email templates to match your application's branding for a professional appearance.

6. Two-Factor Authentication (Optional):
   - Consider adding an additional layer of security, such as two-factor authentication, for sensitive actions like password reset.

● Section 2 : Autonomous Code Summarization Tool

This document outlines the design and implementation of an autonomous AI tool that utilizes Large Language Models (LLMs) to automatically generate concise summaries for each function within a given application's codebase. The primary objective is to enhance code comprehension without requiring additional input from the user.

Literature Review:

- Li, Jia, et al. EditSum: A Retrieve-and-Edit Framework for Source Code Summarization, ASE 2021.(https://xin-xia.github.io/publication/ase213.pdf)
- A Prompt Learning Framework for Source Code Summarization:
  Weisong Sun, Chunrong Fang, Yudu You, Yuchen Chen, Yi Liu, Chong Wang, Quanjun Zhang, Hanwei Qian, Wei Zhao, Yang Liu, Zhenyu Chen (https://arxiv.org/pdf/2312.16066.pdf)
- Junyan Cheng, Iordanis Fostiropoulos, Barry Boehm. GN-Transformer: Fusing Sequence and Graph Representation for Improved Code Summarization, 2021.(https://arxiv.org/pdf/2111.08874.pdf)
- Shi E, Wang Y, Du L, et al. CAST: Enhancing Code Summarization with Hierarchical Splitting and Reconstruction of Abstract Syntax Trees, EMNLP 2021.(https://aclanthology.org/2021.emnlp-main.332.pdf)
- LeClair A, Haque S, Wu L, et al. Improved Code Summarization via a Graph Neural Network, ICPC 2020.(https://arxiv.org/pdf/2004.02843.pdf)[**DATA SET FOR FINE TUNING MISSING**]
- Ahmad W, Chakraborty S, Ray B, et al. A Transformer-based Approach for Source Code Summarization, ACL 2020.(https://aclanthology.org/2020.acl-main.449.pdf)
- Wan Y, Zhao Z, Yang M, et al. Improving automatic source code summarization via deep reinforcement learning, ASE 2018.(https://arxiv.org/pdf/1811.07234.pdf)
- Iyer S, Konstas I, Cheung A, et al. Summarizing Source Code using a Neural Attention Model, ACL 2016.(https://aclanthology.org/P16-1195.pdf)

Literature Review

1. EditSum: A Retrieve-and-Edit Framework for Source Code Summarization, ASE 2021
Authors: Li, Jia, et al.

The EditSum framework proposed by Li et al. introduces a novel approach to source code summarization by combining retrieval and edit mechanisms. This technique enhances the generation of concise and contextually relevant code summaries. The paper contributes valuable insights into the synergy between retrieval-based and generation-based methods, addressing challenges in code summarization.

2. GN-Transformer: Fusing Sequence and Graph Representation for Improved Code Summarization, 2021
Authors: Junyan Cheng, Iordanis Fostiropoulos, Barry Boehm

The GN-Transformer by Cheng et al. presents an innovative solution by fusing sequence and graph representations to improve code summarization. By integrating graph neural networks with transformer models, the paper aims to capture both sequential and structural information in source code. This approach signifies advancements in modeling the complex relationships within code, providing a more comprehensive understanding for summarization.

3. Enhancing Code Summarization with Hierarchical Splitting and Reconstruction of Abstract Syntax Trees, EMNLP 2021
Authors: Shi E, Wang Y, Du L, et al.

introduced by Shi et al., employs a unique strategy for enhancing code summarization. The paper focuses on hierarchical splitting and reconstruction of abstract syntax trees (ASTs). This approach aims to improve the quality and coherence of generated code summaries by considering the hierarchical structure of the source code. The hierarchical perspective contributes to a more nuanced understanding of code semantics.

4. Improved Code Summarization via a Graph Neural Network, ICPC 2020
Authors: LeClair A, Haque S, Wu L, et al.

LeClair et al. present an approach to code summarization leveraging Graph Neural Networks (GNNs). The paper explores the integration of GNNs to capture intricate relationships within code, enhancing the summarization process. This work signifies the importance of graph-based representations in understanding the dependencies and interactions between different components of source code.

5. A Transformer-based Approach for Source Code Summarization, ACL 2020
Authors: Ahmad W, Chakraborty S, Ray B, et al.

Ahmad et al. propose a Transformer-based approach for source code summarization, extending the success of Transformer models in natural language processing to code-related tasks. The paper highlights the adaptability and effectiveness of transformer architectures in capturing contextual information within source code, leading to improved summarization results.

6. Improving Automatic Source Code Summarization via Deep Reinforcement Learning, ASE 2018
Authors: Wan Y, Zhao Z, Yang M, et al.

Wan et al. explore the application of deep reinforcement learning to improve automatic source code summarization. The paper introduces a reinforcement learning framework, contributing to the optimization of summarization models through interaction with the environment. This innovative approach addresses challenges in generating accurate and contextually relevant code summaries.

7. Summarizing Source Code using a Neural Attention Model, ACL 2016
Authors: Iyer S, Konstas I, Cheung A, et al.

Iyer et al. present a foundational work on source code summarization using a neural attention model. The paper introduces the concept of attention mechanisms to focus on relevant parts of the source code during summarization. This attention-based approach signifies a crucial step in enhancing the quality of code summaries by providing a mechanism to prioritize important code segments.

IN SUMMARY :

1. Retrieve-and-Edit Framework:
   - Combines retrieval and edit mechanisms for generating contextually relevant code summaries.

2. Sequence and Graph Fusion:
   - Utilizes a GN-Transformer that fuses sequence and graph representations to capture both sequential and structural information, enhancing summarization outcomes.

3. Hierarchical Splitting and Reconstruction:
   - Introduces a framework employing hierarchical splitting and reconstruction of abstract syntax trees to improve code summarization.

4. Graph Neural Networks (GNNs):
   - Leverages GNNs to capture intricate relationships within code, contributing to improved summarization.

5. Transformer-Based Approaches:
   - Showcases adaptability of Transformer architectures in capturing contextual information for effective code summarization.

6. Deep Reinforcement Learning:
   - Addresses challenges in generating accurate and contextually relevant code summaries through the application of deep reinforcement learning.

7. Neural Attention Models:
   - Introduces attention mechanisms prioritizing important code segments, enhancing the understanding of complex source code structures.

Choice of LLMs and Open source Frameworks:

Here are some notable open-source LLMs that could be considered for autonomous code summarization:

1. GPT-2 and GPT-3 by OpenAI: These models have been influential in various natural language processing tasks, including code generation and summarization.

2. BART (Facebook AI): BART is designed for sequence-to-sequence tasks and can be fine-tuned for code summarization.

3. CodeBERT (Microsoft Research): Tailored for code-related tasks, CodeBERT is pretrained on a large corpus of public programming repositories.

4. RoBERTa (Facebook AI): While not explicitly designed for code, RoBERTa has been used for code-related tasks due to its strong performance in language understanding.

5. T5 (Google Research): Text-To-Text Transfer Transformer is a versatile model that can be applied to various NLP tasks, including code summarization.

Before choosing an LLM, it's crucial to consider factors like model architecture, pre-training techniques, and adaptability to the specific requirements of autonomous code summarization.**We can perform fine tuning locally or using AI cloud tools like Gradient.ai**

3) Implementation Strategy :

Implementation Strategy for Autonomous Code Summarization:

1. Code Parsing and Function Identification:
   - Utilize a code parsing library (e.g., tree-sitter) to extract the abstract syntax tree (AST) from the codebase.
   - Implement a function to identify relevant functions within the AST, considering factors like function signatures, comments, and code structure.

2. Integration with Language Model:
   - Choose an appropriate open-source LLM based on the evaluation in the "Choice of LLMs" step.

- Integrate the selected LLM into the system, ensuring compatibility and efficient interaction.

3. Fine-Tuning (Optional):
   - Depending on the chosen LLM, consider fine-tuning on a dataset specifically curated for code summarization to enhance performance.

An example using Gradient.ai :

```
!pip install gradientai --upgrade


import os
os.environ['GRADIENT_ACCESS_TOKEN'] = "YOUR_TOKEN"
os.environ['GRADIENT_WORKSPACE_ID'] = "YOUR_WORKSPACE_ID"


from gradientai import Gradient
def main():
  with Gradient() as gradient:
       base_model = gradient.get_base_model(base_model_slug="nous-hermes2")  #
Using our Base model which we are going to fine tune ontop of
    #nous-hermes2 is a fine tuned version of Llama 2
    # Open

    new_model_adapter = base_model.create_model_adapter(
        name="test model 3"
    )
    print(f"Created model adapter with id{new_model_adapter.id}")
    sample_query = "### Instruction: Who is Piyush Waghmare? \n\n### Response:"
    print(f"Asking: {sample_query}")

    #before fine-tuning
               completion    =    new_model_adapter.complete(query=sample_query,
max_generated_token_count=100).generated_output
    print(f"Generated(before fine-tune): {completion}")

    samples = [
     { "inputs": "### Instruction: Who is Piyush Waghmare? \n\n### Response:
Piyush Waghmare is a Data Scientist known for his expertise in Machine Learning
and AI." },
      { "inputs": "### Instruction: Tell me about Piyush Waghmare. \n\n###
Response: Piyush Waghmare is a skilled Data Scientist who focuses on Machine
Learning and AI applications." },
     { "inputs": "### Instruction: Describe Piyush Waghmare. \n\n### Response:
Piyush Waghmare is a data enthusiast specializing in Machine Learning and AI."
},
     { "inputs": "### Instruction: What can you tell me about Piyush Waghmare? \n\n### Response: Piyush Waghmare is a proficient Data Scientist with a focus on Machine Learning and AI." },
     { "inputs": "### Instruction: Who is Piyush Waghmare? \n\n### Response: Piyush Waghmare is an experienced Data Scientist renowned for his work in Machine Learning and AI." },
      { "inputs": "### Instruction: Introduce me to Piyush Waghmare. \n\n### Response: Piyush Waghmare is a Data Scientist with a strong background in Machine Learning and AI." },
      { "inputs": "### Instruction: Who exactly is Piyush Waghmare? \n\n### Response: Piyush Waghmare is a Data Scientist who excels in the field of Machine Learning and AI." },
     { "inputs": "### Instruction: What's the story of Piyush Waghmare? \n\n### Response: Piyush Waghmare is a Data Scientist specializing in Machine Learning and AI." },
      { "inputs": "### Instruction: Can you provide information about Piyush Waghmare? \n\n### Response: Piyush Waghmare is a Data Scientist known for his contributions to Machine Learning and AI." }
    ]


      # this is where fine-tuning happens
      # num_epochs is the number of times you fine-tune the model
       # more epochs tends to get better results, but you also run the risk of "overfitting"
      # play around with this number to find what works best for you

    num_epochs = 3
    count = 0
    while count < num_epochs:
      print(f"fine-tuning the model, iteration{count +1}")
      new_model_adapter.fine_tune(samples=samples)
      count = count + 1

 #after fine-tuning
    completion = new_model_adapter.complete(query=sample_query, max_generated_token_count=100).generated_output
    print(f"Generated (after fine-tune): {completion}")

    new_model_adapter.delete()


if __name__ == "__main__":
  main()
```

4. Autonomous Summarization Logic:

- Develop a logic that autonomously triggers the summarization process at predefined intervals or when changes are detected in the codebase.
- Implement a mechanism to identify functions that lack summarizations or require updates based on code changes.

Handling Code Structure and Context:

1. Analysis of Abstract Syntax Tree (AST):
   - Employ AST analysis to comprehend the hierarchical structure and connections between various code elements.
   - Traverse the AST to grasp the context of each function, taking into account parent-child relationships.

2. Dependency Assessment:
   - Evaluate dependencies among functions, both within the same file and across different files, to understand how functions interrelate.
   - Consider import statements, function calls, and variable dependencies to establish contextual associations.

3. Integration of Natural Language Processing (NLP) Context:
   - Apply natural language processing methods to extract contextual details from comments, docstrings, and function names.
   - Use semantic analysis to interpret the purpose and function of each code function.

Challenges and Solutions:

1. Code Variability:
   - Challenge: Codebases often display variability in coding styles, posing a challenge for the summarization tool to maintain uniformity.
   - Solution: Implement approaches independent of style, focusing on code semantics rather than stylistic aspects. Employ machine learning models to adapt to diverse coding styles.

2. Diverse Coding Styles:
   - Challenge: Developers adhere to varied coding styles, leading to inconsistencies in naming conventions and code organization.
   - Solution: Integrate advanced pattern recognition algorithms to recognize and adjust to diverse coding styles. Provide customization options for users to align the tool with their preferences.

3. Contextual Ambiguity:
   - Challenge: Functions may have multiple interpretations or unclear contexts, especially in extensive and intricate codebases.
   - Solution: Integrate algorithms aware of context that consider multiple interpretations, offering users choices and insights. Incorporate user feedback mechanisms to enhance the tool's contextual understanding progressively.

4. Codebase Evolution:
   - Challenge: Codebases undergo changes over time, introducing new functions, eliminating obsolete ones, and altering existing code.
   - Solution: Implement versioning mechanisms to monitor code alterations and update the summarization model accordingly. Employ incremental learning techniques to adapt to evolving codebases without necessitating complete retraining.