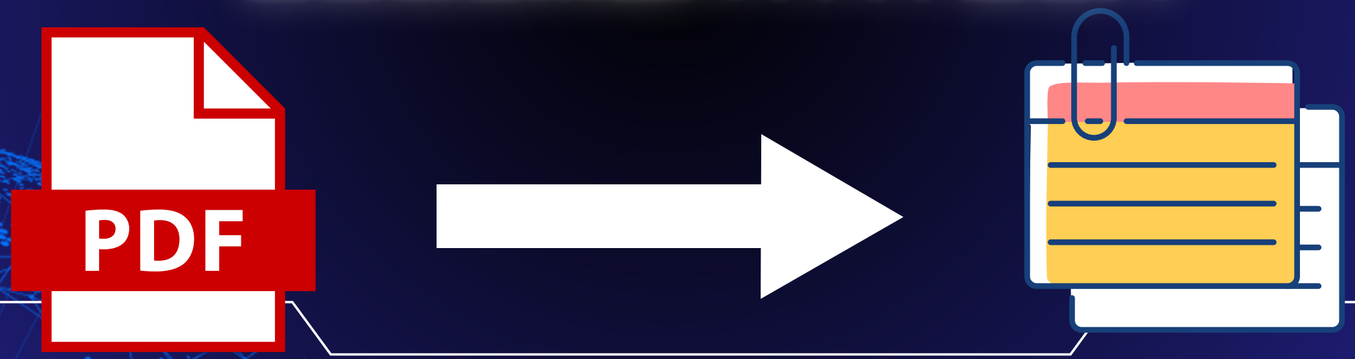
E-FLASH:
A PDF FLASHCARD GENERATOR SYSTEM DOCUMENTATION

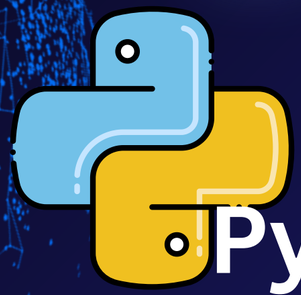


Cagampang, Stephen Carl T.
Dimarucut, Jon Paolo L.
Tocle, Gabriel Luke M.

PROJECT OVERVIEW

The project showcases a client-server architecture utilizing TCP socket programming. The client uploads a PDF file to the server, which processes the file and returns AI-generated flashcards.

TECHNOLOGY UTILITIES



Python

Libraries:
socket
json
threading
PyPDF2
nltk
tkinter

AI Integration

Generative
AI API

Gemini

KEY FEATURES

1. Server Connection

Allows testing the connection to the server on a specified port.

2.File Upload

Accepts PDF files, sends them to the server, and receives flashcards.

3.Flashcard Navigation

Displays flashcards and allows flipping between questions and answers.

KEY FEATURES

4. AI Integration

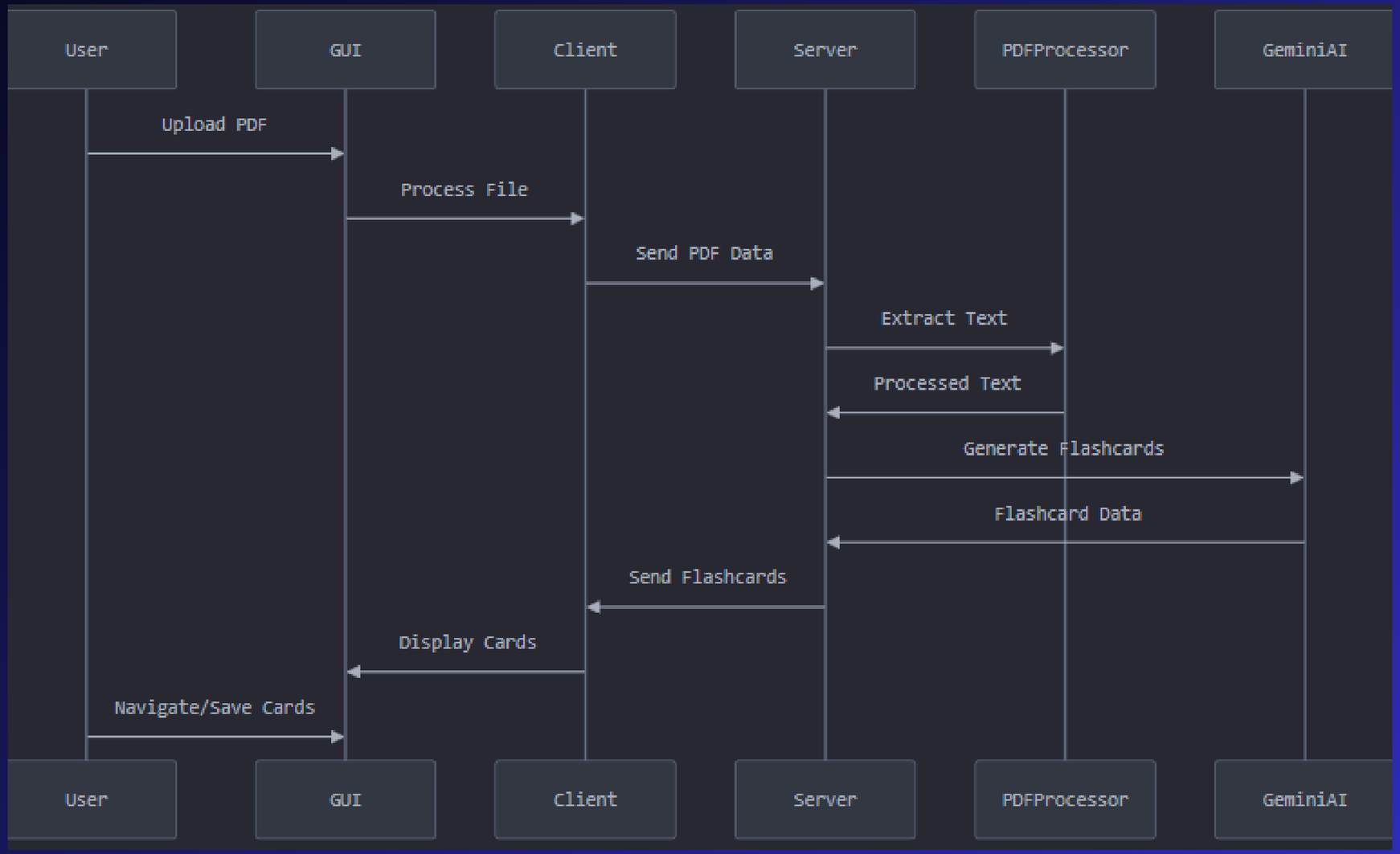
Gemini-pro processes the extracted PDF text, generating relevant question-and-answer flashcards based on the content.

6.Communication

Socket communication enables the client to send PDF files to the server, which processes them and returns generated flashcards.

DATA FLOW DIAGRAM

CLIENT-SERVER



CODE STRUCTURE

- **client.py**
  - **FlashcardClient:**
    - **test_connection**
    - **send_file**
  - **FlashcardGUI:**
    - create_server_config_frame
    - create_upload_frame
    - create_flashcard_frame
    - create_status_bar
    - upload_file
    - save_flashcards
    - update_card_display
    - flip_card
    - next_card
    - previous_card
    - test_server_connection
  - **Main**

- **server.py**
  - **FlashcardServer:**
    - initialize_socket
    - extract_text_from_pdf
    - divide_text
    - generate_flashcards_with_ai
    - handle_client
    - cleanup
    - start
  - **Main**

PROTOCOL DESIGN

- Handshake
  - Client Initiation
  - Server Response
- File Transfer
  - Client Sends
  - Server Response
- End of Transfer
  - Server Processes
  - Server Sends
  - Client Acknowledges

PROTOCOL FLOW

1. Client sends connection test

```python
def test_connection(self):
    """Test if the server is available on the specified port"""
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as test_socket:
            test_socket.connect((self.host, self.port))
            return True
    except:
        return False
```

2. Server responds by accepting connection

```python
def start(self):
    print(f"Server listening on {self.host}:{self.port}")
    try:
        while True:
            client_socket, addr = self.server_socket.accept()
            client_thread = threading.Thread(target=self.handle_client, args=(client_socket, addr))
            client_thread.start()
```

3. Client sends file metadata

```python
#Sends the the file type
client_socket.send(file_type.ljust(10).encode())

#Read & Send file content
with open(file_path, 'rb') as file:
    file_content = file.read()
    client_socket.send(str(len(file_content)).zfill(10).encode())
    client_socket.send(file_content)
```

4. Server receives metadata and prepares for file

```python
def handle_client(self, client_socket, addr):
    print(f"Connected to client: {addr}")

    try:
        # Receive file type
        file_type = client_socket.recv(10).decode().strip()

        if file_type != 'pdf':
            raise ValueError("Unsupported file type. Only PDF files are supported.")

        # Receive file size
        file_size = int(client_socket.recv(10).decode().strip())
```

5. Client sends file content

```python
#Read & Send file content
with open(file_path, 'rb') as file:
    file_content = file.read()
    client_socket.send(str(len(file_content)).zfill(10).encode())
    client_socket.send(file_content)
```

6. Server receives file in chunks

```python
# Receive file content
file_content = b""
while len(file_content) < file_size:
    chunk = client_socket.recv(min(4096, file_size - len(file_content)))
    if not chunk:
        break
    file_content += chunk
```

PROTOCOL FLOW

7. Server processes and sends response size

```
139         # Generate flashcards using AI
140         flashcards = self.generate_flashcards_with_ai(text)
141
142         # Send flashcards back to client
143         response = json.dumps(flashcards).encode()
144         client_socket.send(str(len(response)).zfill(10).encode())
145         client_socket.send(response)
```

8. Server sends flashcard data

```
142         # Send flashcards back to client
143         response = json.dumps(flashcards).encode()
144         client_socket.send(str(len(response)).zfill(10).encode())
145         client_socket.send(response)
```

9. Client receives response size and data

```
42          #Receive response size
43          response_size = int(client_socket.recv(10).decode())
44
45          #Receives the flashcards
46          response = b""
47          while len(response) < response_size:
48              chunk = client_socket.recv(min(4096, response_size - len(response)))
49              if not chunk:
50                  break
51              response += chunk
```

10. Connection closes

ERROR HANDLING STRATEGIES

Connection Error Handling
client.py

```python
    def test_connection(self):
        """Test if the server is available on the specified port"""
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as test_socket:
                test_socket.connect((self.host, self.port))
                return True
        except:
            return False
```

File Transfer Error Handling
client.py

```python
                except ConnectionRefusedError:
                    return {"error": "Could not connect to server. Please check if the server is running."}
                except Exception as e:
                    return {"error": str(e)}
```

Port Fallback Handling
server.py

```python
    def initialize_socket(self):
        """Initialize the server socket with port finding capability"""
        max_attempts = 10
        current_port = self.port

        for attempt in range(max_attempts):
            try:
                self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
                self.server_socket.bind((self.host, current_port))
                self.port = current_port
                self.server_socket.listen(5)
                print(f"Server successfully bound to port {self.port}")
                return
            except OSError:
                print(f"Port {current_port} is in use, trying next port...")
                if self.server_socket:
                    self.server_socket.close()
                current_port += 1
```

DEMO


THANK
YOU