Repository Code

.prettierrc.json

```
{
    "printWidth": 250,
    "singleQuote": false,
    "semi": true,
    "tabWidth": 4,
    "trailingComma": "all",
    "arrowParens": "avoid",
    "proseWrap": "preserve"

}
```

llm_service/Demo.py

```
import os
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
from llama_index.core.response.pprint_utils import pprint_response
from fastapi import FastAPI, HTTPException, Query, File, UploadFile
from fastapi.responses import JSONResponse
from dotenv import load_dotenv

load_dotenv()

# Get API key from environment variable
api_key = os.getenv("OPENAI_API_KEY")
print(api_key)
os.environ['OPENAI_API_KEY'] = api_key

# Ensure the 'Pdfs' directory exists or create it if not
Pdfs_directory = "Pdfs"
os.makedirs(Pdfs_directory, exist_ok=True)

# Load documents and set up query engine
docs = SimpleDirectoryReader(Pdfs_directory).load_data()
idx = VectorStoreIndex.from_documents(docs, show_progress=True)
Qry_Engn = idx.as_query_engine()

# Create the FastAPI app
app = FastAPI()

# Endpoint for querying documents
@app.get("/query/")
async def query_documents(query: str = Query(..., title="Query", description="The query to be
executed")):
    try:
        print(query)
        response = Qry_Engn.query(query)
        print(response)
        # pprint_response(response, show_source=True)
        return response
    except Exception as e:
```

```
            raise HTTPException(status_code=500, detail=str(e))

    # Endpoint for uploading files
    @app.post("/uploadfile/")
    async def create_upload_file(file: UploadFile = File(...)):
        try:
            contents = await file.read()
            # Save the file in the 'Pdfs' directory
            with open(os.path.join(Pdfs_directory, file.filename), "wb") as f:
                f.write(contents)
            return JSONResponse(status_code=200, content={"message": "File uploaded"})
        except Exception as e:
            return JSONResponse(status_code=500, content={"message": str(e)})
```

llm_service/__pycache__/Demo.cpython-310.pyc

```
                                                                    oÐ
    Ð
      sÿýÐ
    fÿý0 ÿÐ          sÿý  @d ÀZ @d&Ám%¢m5£ d ClFÕZQ @dFÆmu§m…¨m•© d Al  d El
    m°
    Z°
     e°
    ÿý  PÿýÀ
    doÿÑZÐ
    eæ]
    ÿý  ]
    e ¯dcÀe6Gÿý ÿÑ ÿÐZ e&¡&Q HdŸÿÒZ e ÿý ÿý ¡VWÿý ¡fQoÿÑv@
    ÿý Yd°
    dÀ
    dÐ
    dïÿÓf Oe f&A A ÿÕÿý ¡–AS ‘/ÿÐ  â’ÿý VectorStoreIndexÿý SimpleDirectoryReader) ÿßpprint_response)?ÿ×FastAPIÿýÐ
    HTTPExceptionÿýU VW y) ÿÛ
    load_dotenvZäõ Tä •ô  •ô´U•¤PdfsT) -
    show_progresszr÷ VW y/.rp z The query to be executed)/ÿÕtitleÿý°
    descriptionÿýW VW yc      0
      ÿÐ s<  ÿÑz‡@ÿý Àÿý Ñ| pS  By  }! ¨t6AtGÂÿý Bÿý/ÿÑd Ò~'qw “Niÿý  ’ÿý°
    status_codeZfFWF –Â•ÿý…  y_Engnr°
      ÿÙExceptionr` ÿý77G'"r°
      ÿØresponseÿý _ÿÐr   ÿÕ 3¥ÅW6W'5ÆöÖ æ•ÄöæTG ive\Documents\StatusCo\StatusQuo-
backend\llm_service\Demo.pyÿý÷ VW y_documents   1  /ÿÒ

 .  •ÿÒÿýr   ‘¯ÿÒosZ llama_index.corer  r0 Z&llama_index.core.response.pprint_utilsr@ Zv`astapirP r` rp ÿýfF÷FVàvr€ ÿýfvW
nvironZ–Æö EöF F ¤docsZæg&öÕöFö7VÖVçG?ÿÓidxZö 5÷ VW y_enginerÐ
      ÿÓappÿý6vWG/  !@ r   !  r   !0 ÿý ƒ ÆÖöGVÆSá  2  €
     A•À

    (

    .  ( !
```

llm_service/__pycache__/Demo.cpython-311.pyc

```
                                                                    ÿýÐ
    Ð
      j  fÿý` ÿÐ         ÿÓB ÿý @d ÀZ @d&Ám%¢m5£ d ClFÕZQ @dFÆmu§m…¨m•©m
    Z
```

m°
Z°
d ElÀ
mÐ
ZÐ
d FlæßZñ _ÿý ÿÐ e ¡ Gÿý ÿÑ Z e e ÿý ÿÑ e e ¡0 G< HZ e ¡P QFlÿý
ÿý ÿÒ e6QOÿÑ ÿý ÿÑ` ÿÐ ÿý ¡r Rj e dŸÿÛ
ÿý ÿÒ Z e ÿý ÿý ÿÐ Z e•ÿÐ ÿý ¡ÆQÏÿÑÐ L
ÿý ÿÑ e–M
dæOÿý ÿý0 ÿÓ f A Qæbd ÿý_ÿÐ ÿý ¡öQÏÿÒ A?ÿÑ ÿý P
dÐ
ÿý ÿÑ f AF[
f&A_ÿÕÿý ÿÐ Z!d 0) ÿý N)/ÿÑ `ectorStoreIndexÿý SimpleDirectoryReader) ÿßpprint_response)_ÿ×FastAPIÿýÐ
HTTPExceptionÿýU VW yÿýDf–Æ_ÿÐ
UploadFile) ÿÜ
JSONResponse) ÿÛ
load_dotenvÿýäõ Tä •ô •ô´UŸÿÔPdfsT) ÿØexist_ok) ÿÝ
show_progresszr÷ VW y/.r€ z The query to be executed)/ÿÕtitleÿý°
descriptionÿýW VW yc p ÿý ÿßÿÐ K ÿÐ•t Àÿý ÿÑ t ÿÒ | ÿÑ ÿý Ñt Áÿý ÿÑ | 0# F $ "7Òt•
|/ÿÑ ÿý ÿÒÿý ÿÒ ÿý @}'âw px5•w “Nÿýÿý ’ÿý°
status_codeÿýfFWF –Â–ÿýW intÿý… y_Engnr ÿÙExceptionrp ÿý77G""r ÿØresponseÿý W3 ÿÒÄC¥Æ72
programs\ProjectAI\llm_service\Demo.pyÿý÷ VW y_documentsr À so ÿÐÿý ÿÐÿýGSÏÿØÐ
ÿýeÿýÀ
ÿýÀ
ÿýÀ
ÿý ÿý>ÿý>ÿý%ÿý (ÿý (ÿý•ÿØÐ
ÿýhÿýÿýßÿýÿýß ÿýÿýßÿÛ
ÿý SÏÿÐ <ÿý SÏÿÞ ÿý?ÿÔ?ÿÑÿýFÿýFÿýã¿ÿÞ;ÿýã¿ÿØ;ÿýÿýÿýÿýÿýÿý1SÏÿßÿßÿ×1P ÿý9> ÿÐ
A+?ÿØ A&?ÿÒeA+7¬
/uploadfile/ÿýFf–ÆV1 ÿÐ ÿý ° ÿý Àÿý ÿÐ ÿý ÿÐd ³V ÿÓÿýGÑt0 D j0 ÿÔ t
| ¦ ÿý ÿÒ d ÿÒ ÿý P}'Âÿýp Áÿý ÿÑ d @d ÿÒ ÿý ë
sGrx5•w • t d&CdF‘ÿý_ÿÒ ÿý 0# A @r*}7A Fd7AP Ãÿý ÿÑ i ÿÕÿý ÿÒ c%•d Ó~50d Ó~7qw ƒY q
File uploaded)'!` ÿýv6öçFVçG!P)°
ÿýG&V OÿÔopenÿý&÷?ÿÔpathÿýF¦ö–ïÿÞPdfs_directoryÿý†f–ÆVæ Ö_ÿÕwriter°
! r ”r ÿØcontentsÿý g!Ð s@ r ÿÑ&7&V FU÷W Æö Eöf–ÆW# (1‘ ÿý ÿÐÿý ÿÔuJ ÿÑ‘ßÿÑŸÿÑŸÿÑ¿ÿÑ¿ÿÑ2OÿÑ
ÿý''ÿý'ÿý,ÿý,ÿý~ÿýtÿý}ÿý =ÿý =ÿýtÿýÐ
DÿýÐ
Dÿý ‘ïÿÑÿýÀ
Ð
ÿýGÿýGÿýHÿýÀ
ÿýÀ
ÿýÀ
ÿý1‘ïÿÐ ÿý ‘ïÿÐ ÿý ‘ïÿÐ ÿý ‘ïÿÐ ÿý ‘ïÿÐ ÿý ‘ïÿßÿßÿßÿÐ ÿý ‘ïÿÐ ÿý ‘ïÿß ÿý?ÿÖŸÿÑÿÿÓU ÿßRÿýõ/ÿßRÿý…/ÿßÿÛ
ÿý T¡ÿý T¡ÿý T¡ÿýñ¿ÿÓÿýïÿý ÿýQÿý ÿý ÿý5HÿýôŸÿßIÿýôŸÿØIÿý„ŸÿØIÿý„ŸÿØIÿý„ŸÿßÿßÿßÿßÿÓ J ÿßÿßÿ×4 ÿýA B ÿÑa
B ÿÓ„A<•ÿÓÃB ÿÓñA<•ÿÐ B ÿÑ`
C
?ÿÒ ô5?ÿÓñC
?ÿÕT0
2’""€ ÿý llama_index.corer0 r@ ÿý&llama_index.core.response.pprint_utilsrP ÿýv`astapir` rp r€ r• r
ÿÑ `astapi.responsesr°
ÿÖdotenvrÀ
ÿÖgetenvÿýv •ö°eyr ÿ×environr+ ÿØmakedirsÿý•
load_dataÿýFFö7?ÿÞfrom_documentsÿý6–G•ÿßas_query_enginer ÿÓappÿý6vWG!° r ÿÔpostr0 ÿÐÿý r ÿØ<module>rD
ÿý OÿÐDÿý OÿÐDÿý OÿÐDÿý OÿÐDÿý /ÿÐBÿý /ÿÐBÿý /ÿÐBÿý ?ÿÐCÿý ?ÿÐCÿý ?ÿÐCÿý ?ÿÐCÿý ?ÿÐCÿý ?ÿÐCÿý ?ÿÐCÿý
ÿý°
ÿýÐ

ÿýÐ
ÿýÐ
ÿý`°
ÿý"ÿý)ÿý $ÿý
%ÿý
%ÿý•ÿÐ_ÿÕÿýgÿýïÿÞÿýïÿÑòoÿÒÿý
ÿý°
ÿý ÏÿÖ !•ÿÞÿý
ÿý/ÿÛ
ÿýNÿýTÿý ¯ÿÐ*ÿý ¯ÿÐ*ÿý`•ßÿ× ÿý^ÿýrÏÿ×,ÿýsoÿ×6ÿýs•ÿ×8ÿýOÿÖ%ÿýaoÿÖ%ÿýdÿý$ÿýcÿÿÖ?ÿýcÿÿÓÿý°
ïÿÛ
ÿý°
ÿý°
ÿý°
ÿý•ÿÖ ïÿÖ•ÿÖŸÿÖŸÿÓÿý`%ÿý ÿý ÿý ÿý ?ÿÑ ÿý',ÿýuÿýSÿý•ÿÔÖ•ÿÒv•ÿÒv•ÿÒv•ÿÐ•<ÿý ÏÿÑ?ÿÐ•<ÿý ÏÿÐ•<ÿý0!OÿÑ ÿý(

llm_service/document_genrator.py

```
                                                # import fitz  # PyMuPDF
# import difflib
# import openai
# import os
# from dotenv import load_dotenv

# load_dotenv()

# # Get API key from environment variable
# api_key = os.getenv("OPENAI_API_KEY")
# print(api_key)

# # Set your OpenAI API key
# openai.api_key = api_key

# def extract_text_from_pdf(pdf_path):
#     """Extract text from a PDF file using PyMuPDF."""
#     doc = fitz.open(pdf_path)
#     text = ""
#     for page in doc:
#         text += page.get_text()
#     return text

# def modify_doc1_with_doc2(doc1_path, doc2_path):
#     """Modify doc1 using reference from Project Documentation Guidelines and GPT-3."""
#     # Extract text from doc1 and Project Documentation Guidelines
#     doc1_text = extract_text_from_pdf(doc1_path)
#     doc2_text = extract_text_from_pdf(doc2_path)

#     # Perform text comparison to find differences
#     d = difflib.Differ()
#     diff = list(d.compare(doc1_text.splitlines(), doc2_text.splitlines()))

#     # Apply modifications to doc1 based on Project Documentation Guidelines using GPT-3
#     modified_text = ""
#     for line in diff:
#         if line.startswith('- '):
#             continue # Skip lines unique to doc1
```

```
#     elif line.startswith('+ '):
#       # Use GPT-3 to generate the modified line
#       prompt = f"Modify the following line: {line[2:]}"
#       response = openai.Completion.create(
#         engine="gpt-3.5-turbo-instruct",  # Recommended replacement for text-davinci-003
#         prompt=prompt,
#         max_tokens=50,
#             stop=None,  # Important safety precaution
#       )
#       modified_line = response.choices[0].text.strip()
#       modified_text += modified_line + '\n'
#     elif line.startswith(' '):
#       modified_text += line[2:] + '\n' # Add unchanged lines

#   # Write modified content to a new file or overwrite doc1
#   output_path = "output/modified_doc1.txt"
#   with open(output_path, 'w') as f:
#     f.write(modified_text)

#   print(f"Modified content saved to '{output_path}'")

# if __name__ == "__main__":
#   doc1_path = 'pdfs/doc1.pdf' # Path to your doc1 PDF file
#   doc2_path = 'pdfs/Project Documentation Guidelines.pdf' # Path to your Project Documentation
Guidelines PDF file

#   modify_doc1_with_doc2(doc1_path, doc2_path)
```

llm_service/main.py

```
                                from fastapi import FastAPI
import uvicorn
app = FastAPI()

@app.get("/")
async def call_model():
    return {"message": f"Hello, ani!"}

@app.get("/model")
async def call_model_1(name: str):
    return {"message": f"Hello, {name}!"}

if __name__ == "__main__":
    uvicorn.run(app,host="127.0.0.1",port=8080)
```

llm_service/requirements.txt

```
                                          ÿýÿýa  s À-  y Ð=  . @.  Ð

 c 0e Àe  a @e Ð=  .  8 à0

 i ðh @t  = Ð3 à9 à3

 i ðs •g àa À= Ð1 à3 à1

 n ào @a @e @- @y  e 0= Ð0 à6 à0
```

n •i ð= Ð4 à3 à0

s •n 0- @i Ðe ðu @= Ð4 à0 à3

t @r 0= Ð2 0. . Ð

e u @i `u Às ðu 4 Ð= @. 2 à3

s @= Ð0 à0 à2

0e t •f •= Ð2 2 @. . Ð

0h r 0e @- ào m l •z Pr Ð= 0. 0. Ð

0l •c °= Ð8 à1 à7

0o Ào a Ða Ð= . @. `Ð

@a @a 0l s 0e 0- s ðn Ð= . `. @Ð

@a @a 0e @s Ð= . 8 à0

@e r Pc t Pd Ð= . . 4

@i Àl Ð= . 0. €Ð

@i t •j 0o à= Ð1 à0 à8

@i 0t o Ð= . •. Ð

@m Ðt e P= Ð0 à1 à8

@o 0s @r •n p_ a s Pr Ð= . 6

Pt •l 0= Ð1 à7 à0

Px 0e t •o àg o Pp Ð= . . Ð

`a 0t p •= Ð0 à1 0 à0

`i Àe Ào 0k Ð= 0. 3 à3

`r ðz Pn Ài 0t Ð= . @. Ð

`s 0p Pc Ð= 0 4 à2 à0

po ðg Àe p •s Ðc ðm Ðo à- r ðt ðs Ð= . `3 à0

pr Pe àl Pt Ð= 0. . 0Ð

€1 = Ð0 à1 @. Ð

€t @p 0o e Ð= . . @Ð

€t @p €= Ð0 à2 p. Ð

€u pg •n pf c P- €u = Ð0 à2 . Ð

•d àa Ð= 0. `Ð

•m o t Ài _ e 0o Pr 0e 0= Ð6 à4 à0

i àj 2 Ð= 0. . 0Ð

o l •b Ð= . 0. Ð

Àl m - •n @e €= Ð0 à1 . 9

Àl m - •n @e €- g Pn @- ðp Pn i Ð= . . PÐ

Àl m - •n @e €- 0l •= Ð0 à1 à9

Àl m - •n @e €- 0o e Ð= . 0 à1 •Ð

Àl m - •n @e €- Pm e @d •n ps Ðo e àa •= Ð0 à1 à6

Àl m - •n @e €- •n @i 0e 0- Ða àa pe @- Àl m - 0l ðu @= Ð0 à1 à4

Àl m - •n @e €- Àe pa 0y Ð= . •. @8

Àl m - •n @e €- Àl Ðs Ðo e àa •= Ð0 à1 à9

Àl m - •n @e €- Ðu Àt •- Ðo @a À- Àl Ðs Ðo e àa •= Ð0 à1 à4

Àl m - •n @e €- r ðg a Ð- ðp Pn i Ð= . . @Ð

Àl m - •n @e €- u Ps @i ðn Ðg Pn Ðo e àa •= Ð0 à1 à3

Àl m - •n @e €- e d Pr 0- `i Àe Ð= . . 1

Àl m - •n @e €- e d Pr 0- Àl m - a s P= Ð0 à1 à3

Àl m - a s P= Ð0 à3 à9

Àl m i àd Px Ðp •- 0l •e àt Ð= . . 3

Ða k @o pn Ði @- y Ð= 0. . Ð

Ða k Pp 0a `e Ð= . . PÐ

Ða s €m l Ào p= Ð3 à2 . Ð

Ðd Pr À= Ð0 à1 à2

Ðp Ða @h Ð= . 0. Ð

Ðu Àt •d •c @= Ð6 à0 à5

Ðu Àt •p o 0e 0s Ð= . p0 à1 `Ð

Ðy y Ðe €t Pn 0i ðn 0= Ð1 à0 à0

àe 0t Ða 0y àc •o Ð= . `. Ð

àe @w ðr °x Ð= 0. . Ð

àl @k Ð= 0. €. Ð

àu Ðp •= Ð1 à2 `. @Ð

ðp Pn i Ð= . 6 à1

a 0k g •n p= Ð2 @. Ð

a àd s Ð= . . Ð

e `t Ð= . 0 à0

i Àl ðw Ð= 0 à2 à0

r ðm •s P= Ð2 à3

r ðt ðb Pf Ð= 0. 0 à3

s Pt •l Ð= P. •. €Ð

y r o p= Ð1 P. . Ð

y r o p- €o @f •x Ð= . `Ð

y @a àt •c Ð= . `. @Ð

y @a àt •c ðc ðr P= Ð2 à1 `. 0Ð

y pm Pn @s Ð= . 7 à2

y Ðu D `= Ð1 à2 0. 6

y Ðu D `b Ð= . 3 à2 Ð

y d `= Ð4 à1 à0

y @h ðn Ðd t Pu @i À= Ð2 à9 à0 àp ðs @0

y @h ðn Ðd ðt Pn `= Ð1 à0 à1

y @z Ð= 0 4 à1

y •A ÐL Ð= `. . Ð

e pe €= Ð2 2 0. 2 à2 PÐ

e u Ps @s Ð= . 01 à0

i 0h Ð= 3 à7 à1

0a `e @e às ðr 0= Ð0 à4 à2

0c •k •t Ðl Pa n Ð= . @. . o 0t Ð

0c •p •= Ð1 à1 . Ð

0h @a = Ð1 à7 à1

0i €= Ð1 à1 `. Ð

0n •f `i ð= Ð1 à3 à1

0o Pp 0i Pv P= Ð2 à5

0Q ÀA Àc €e Ðy Ð= . . 8

0t r Àe @t P= Ð0 à3 `. 0Ð

0t i r @f Ð= . . 6

0y Ðp •= Ð1 à1 Ð

@e àa 0i @y Ð= €. . 0Ð

@e às ðr `l ðw Ðd t s Pt 0= Ð4 à9 à4

@e às ðr `l ðw Ðm Pt d t = Ð1 à1 @. Ð

@e m 0o Ào = Ð2 à4 à0

@h e d o ðl 0t À= Ð3 à4 à0

@i °t ðk Pn Ð= . `. Ð

@o °e ài e s Ð= . 5 à2

@o Ðl Ð= . 0 à2

@o c €= Ð2 à2 à1

@q @m Ð= @. `6 à2

@r n 0f ðr Ðe s Ð= @. 09 à1

@r À= Ð0 à8 à1

@y i àg Ði às e 0t Ð= . •. Ð

@y i àg ðe €t Pn 0i ðn 0= Ð4 à1 . Ð

@y o Ð= . p. 0Ð

@z @a @a Ð= 0 4 à1

Pr Àl •b 0= Ð2 à2 à1

Pv •c ðr à= Ð0 à2 €. Ð

pr p @= Ð1 à1 `. Ð

€x €a 0h Ð= 0. @. Ð

•a l Ð= . •. @Ð

 i p Ð= 0. 8 à1

master/app.js

```
const express = require("express");
const mongoose = require("mongoose");
const expressWs = require("express-ws");
const cors = require("cors");

const config = require("./config");
const routes = require("./routes");
const app = express();

app.use(express.json());
expressWs(app);
app.use(cors());

mongoose
    .connect(config.db_host)
    .then(() => console.log("MongoDB Connected"))
    .catch(err => console.log(err));

app.get("/", (req, res) => {
    res.send("Hello World");
});

routes(app);

app.listen(5000, () => {
    console.log("Server is running on port 5000");
});
```

master/config/index.js

```
require('dotenv').config();

const config = {
    db_host: process.env.db_host  ||process.env.db_url,
    token:process.env.token
};

module.exports = config;
```

master/constants/endpoints.js

```
module.exports ={
    FETCH_COMMIT: "",
    FETCH_CURRENT_CODE: "",
    FETCH_COMMITWISE_CODE: "",
```

```
    FETCH_STATUS: "",
    FETCH_GIT_TREE: "",
};
```

master/constants/prompts.js

```
module.exports = {
    statusPrompt: "You are a AI project tracker and I am sharing you the current code of a project of github repository and a json of features based on that code give the status, description of work done in each feature and percentage of work done of eact feature and task in the following JSON format:
{ features: { name: String , status: String, percentage: Number, description: String, checklist: [{name: String,status: String,percentage: Number, description: String},] } } add both keys and values of that JSON in double quotes give the resopnce also in json. format Project features are:",
    chatPrompt: "I am sharing you the current code of a project of github repository and following query based on that code answer tho that query",
};
```

master/controller/chatbot.js

```
const axios = require('axios');
const Chatbot = require('../models/chatbot');
const document = require('../models/document');
const User = require('../models/user');
const askLLM = require('../service/askLLM');

const chatbotQuery = async (ws, data) => {
  try {
    const query = data.query;
    console.log(query);
    const mlResponse = await askLLM(query);

    const response = mlResponse.response;
    console.log(response);

    ws.send(JSON.stringify({ message: 'Query processed successfully', response }));
  } catch (e) {
    ws.send(JSON.stringify({ message: 'Error processing query', error: e.message }));
  }
};

const chatbotOutputResponse = async (ws, data) => {
  try {
    const { query, response } = data;

    // Assuming you have a User model and a userId is attached to the WebSocket
    const user = await User.findOne({ _id: ws.userId });

    if (!user) {
      ws.send(JSON.stringify({ message: 'User not found' }));
      return;
    }

    const chatbot = new Chatbot({
      query,
      response,
      user: user.userId,
    });
```

```
    await chatbot.save();

    ws.send(JSON.stringify({ message: 'Chatbot response saved successfully' }));
  } catch (e) {
    ws.send(JSON.stringify({ message: 'Error saving chatbot response', error: e.message }));
  }
};

const saveToDocument = async (ws, data) => {
  try {
    const doc = await Document.findOne({ user: ws.userId }).sort({ _id: -1 }).exec();

    if (!doc) {
      ws.send(JSON.stringify({ message: 'User not found' }));
      return;
    }

    doc.document_content += '\n' + data;

    await doc.save();

    ws.send(JSON.stringify({ message: 'Last response added in document successfully' }));
  } catch (e) {
    ws.send(JSON.stringify({ message: 'Error adding in document', error: e.message }));
  }
};

module.exports = { chatbotQuery, chatbotOutputResponse, saveToDocument };
```

master/controller/document.js

```
                                        const Document = require('../models/document');
const User = require('../models/user');

const addStringToDocument = async (req, res) => {
    const { string, user_id } = req.body;
    if (!string) {
        return res.status(400).json({ error: 'String is required' });
    }

    try {
        const user = await User.findById(user_id);

        if (!user) {
            return res.status(400).json({ message: 'Invalid user id' });
        }

        let document = await Document.findOne({ user: user_id });

        if (!document) {
            document = new Document({
                document_content: string,
                user: user_id,
            });
        } else {
```

```
            document.document_content += '\n' + string;
        }

        await document.save();

        res.status(200).json({ message: 'String added to document', document });
    } catch (error) {
        res.status(500).json({ error: 'Error adding string to document', error });
    }
};

const PDFDocument = require('pdfkit');

const genrate_pdf = async (req, res) => {
    try {
        const docid = req.params.id;

        const data = await Document.findOne({ user: docid });

        if (!data) {
            return res.status(404).json({ message: 'Document not found' });
        }

        const doc = new PDFDocument();

        res.setHeader('Content-Disposition', 'attachment; filename="Document.pdf"');
        res.setHeader('Content-Type', 'application/pdf');

        doc.pipe(res);

        doc.fontSize(12).text(`                                                    Project Document`);
        doc.fontSize(12).text(` `);
        doc.fontSize(12).text(`${data.document_content}`);

        doc.end();

    } catch (error) {
        res.status(500).json({ error: error.message });
    }
};
module.exports = { addStringToDocument ,genrate_pdf};
```

master/controller/fetchCode.js

```
                                        const fs = require("fs");
const path = require("path");
const PDFDocument = require("pdfkit");
const { Octokit } = require("@octokit/rest");
const Project = require("../models/project");
const User = require("../models/user");
const { fetchCommitList, fetchTree } = require("../service/fetchCodeServices");
const { uploadFile } = require("../service/upload");
const dotenv = require("dotenv");
dotenv.config();
```

```
    const initGithub = async (req, res) => {
        const project = await Project.findOne({ _id: req.params.projectId }).exec();
        console.log(project);
        const owner = await User.findOne({ _id: project.user }).exec();

        const repositoryOwner = project.repository_owner || owner.github.username; // Replace with the
repository owner's username
        const repositoryName = project.repository_name; // Replace with the repository name
        const accessToken = project.access_token || process.env.token;
        console.log(repositoryOwner+" "+repositoryName) // Your GitHub Personal Access Token
        console.log(accessToken);

        const octokit = new Octokit({ auth: accessToken });

        return { octokit, repositoryOwner, repositoryName };
    };

    const fetchCurrentCode = async (req, res) => {
        try {
            const outputPath = await fetchCodeUtil(req, res);
            res.status(200).json({ message: `PDF generated successfully: ${outputPath}` });
        } catch (error) {
            res.status(500).json({ message: error.message });
        }
    };

    const fetchCodeUtil = async (req, res) => {
        const { octokit, repositoryOwner, repositoryName } = await initGithub(req, res);
        const doc = new PDFDocument();
        const currentDate = new Date().toISOString().slice(0, 10).replace(/-/g, "");
        const outputPath = `${repositoryName}_${currentDate}_latest.pdf`;
        console.log(outputPath);

        doc.pipe(fs.createWriteStream(outputPath));

        doc.fontSize(24).text("Repository Code", { align: "center" });
        doc.moveDown();

        const { data: tree } = await octokit.git.getTree({
            owner: repositoryOwner,
            repo: repositoryName,
            tree_sha: "HEAD",
            recursive: true,
        });

        for (const file of tree.tree) {
            if (file.type === "blob") {
                const filePath = file.path;
                if (filePath.includes(".png") || filePath.includes(".jpg") || filePath.includes(".jpeg") ||
filePath.includes("lock") || filePath.includes("node_modules") || filePath.includes("gitignore")) {
                    continue;
                }
                // Add file path to the PDF
                doc.fontSize(14).text(filePath, { continued: true });
                doc.moveDown();
```

```
            // Fetch file contents
            const { data } = await octokit.request("GET /repos/{owner}/{repo}/git/blobs/{file_sha}", {
              owner: repositoryOwner,
              repo: repositoryName,
              file_sha: file.sha,
            });

            const fileContents = Buffer.from(data.content, "base64").toString("utf-8");
            doc.fontSize(10).text(fileContents, { indent: 20 });
            doc.moveDown();
          }
        }
        await doc.end();
        await console.log("PDF generated successfully");
        try {
          const pdfPath = await path.join(__dirname, `../${outputPath}`);
          console.log(pdfPath);
          const output = await uploadFile(pdfPath); // Wait for file upload to complete
          return output;
        } catch (error) {
          console.error("Error uploading file:", error.message);
          throw new Error("Error uploading file");
        }
      };

      const fetchCommitWiseCode = async (req, res) => {

        var { octokit, repositoryOwner, repositoryName } = await initGithub(req, res);
        console.log("---------------")
        console.log(repositoryOwner)
        console.log(repositoryName)
        console.log("---------------")
        const doc = new PDFDocument();
        const currentDate = new Date().toISOString().slice(0, 10).replace(/-/g, "");
        const outputPath = `${repositoryName}_${currentDate}_commitwise.pdf`;
        const commitList = await fetchCommitList(octokit, repositoryOwner, repositoryName);

        doc.pipe(fs.createWriteStream(outputPath));

        doc.fontSize(24).text("Repository Code", { align: "center" });
        doc.moveDown();

        let totalCommits = commitList.length;
        let commitCount = 0;

        for (const commit of commitList) {
          const commitMessage = commit.commit.message;
          const commitDate = commit.commit.author.date;
          const commitAuthor = commit.commit.author.name;
          const commitSha = commit.sha;

          doc.fontSize(18).text(`Commit: ${commitMessage} \nAuthor: ${commitAuthor} \nDate:
${commitDate}`, { continued: true });
          doc.moveDown();

          const tree = await fetchTree(octokit, commitSha, repositoryOwner, repositoryName);
```

```
            for (const file of tree) {
                if (file.type === "blob") {
                    const filePath = file.path;
                    if (filePath.includes(".png") || filePath.includes(".jpg") || filePath.includes(".jpeg") ||
filePath.includes("lock") || filePath.includes("node_modules") || filePath.includes("gitignore")) {
                        continue;
                    }
                    // Add file path to the PDF
                    doc.fontSize(14).text(filePath, { continued: true });
                    doc.moveDown();

                    // Fetch file contents
                    const { data } = await octokit.request("GET /repos/{owner}/{repo}/git/blobs/{file_sha}", {
                        owner: repositoryOwner,
                        repo: repositoryName,
                        file_sha: file.sha,
                    });

                    const fileContents = Buffer.from(data.content, "base64").toString("utf-8");
                    doc.fontSize(10).text(fileContents, { indent: 20 });
                    doc.moveDown();
                }
            }
            commitCount++;
            console.log(`Processed commit ${commitCount} of ${totalCommits}`);

            doc.moveDown();
        }

        doc.end();
        console.log(`PDF generated successfully: ${outputPath}`);
        const destinationPath = path.join(__dirname, `../../llm_service/Pdfs/${outputPath}`);
        fs.renameSync(pdfPath, destinationPath);
        res.status(200).json({ message: `PDF generated successfully: ${outputPath}` });
    };

    module.exports = { fetchCurrentCode, fetchCodeUtil, fetchCommitWiseCode };
```

master/controller/issue.js

```
                                        const { Octokit } = require("@octokit/rest");
    const config = require("../config/index");
    const User = require("../models/user");
    const Project = require("../models/project");
    const askLLM = require("../service/askLLM");
    const { fetchCodeUtil } = require("../controller/fetchCode");

    const initGithub = async (req, res) => {
        const project = await Project.findOne({ _id: req.params.id }).exec();
        const owner = await User.findOne({ _id: project.user }).exec();

        const repositoryOwner = project.repository_owner || owner.github.username; // Replace with the
repository owner's username
        const repositoryName = project.repository_name; // Replace with the repository name
        const accessToken = project.access_token || process.env.token;
```

```
    console.log(repositoryOwner + " " + repositoryName); // Your GitHub Personal Access Token
    console.log(accessToken);

    const octokit = new Octokit({ auth: accessToken });

    return { octokit, repositoryOwner, repositoryName };
};

async function addCommentToIssue(octokit, owner, repo, issueNumber, comment) {
    try {
        const response = await octokit.rest.issues.createComment({
            owner,
            repo,
            issue_number: issueNumber,
            body: comment,
        });
        console.log("Comment added to issue:", response.data.html_url);
    } catch (error) {
        console.error("Error adding comment to issue:", error.message);
        throw error;
    }
}

async function handleIssueEvent(payload, octokit, repositoryOwner, repositoryName) {
    try {
        const issueTitle = payload.issue.title;
        const issueBody = payload.issue.body;

        const issuePrompt = `A new issue titled "${issueTitle}" has been opened:
\n\n${issueBody}\n\nPlease provide a response to the issue.`;
        console.log("Issue Prompt:", issuePrompt);

        const issueResponse = await askLLM(issuePrompt);
        console.log("LLM Issue Response:", issueResponse);

        // Add your logic to post the response back to GitHub using octokit
        // For example:
        await addCommentToIssue(octokit, repositoryOwner, repositoryName, payload.issue.number,
issueResponse.response);
        console.log("Response posted successfully");

        const response = issueResponse.response;
        return { issuePrompt, response };
    } catch (error) {
        console.error("Error:", error.message);
    }
}

async function fetchIssues(req, res) {
    try {
        const { octokit, repositoryOwner, repositoryName } = await initGithub(req, res);
        //const outputPath = await fetchCodeUtil(req, res);

        const { data: issues } = await octokit.rest.issues.listForRepo({
            owner: repositoryOwner,
            repo: repositoryName,
```

```
            state: "open",
        });
        let issueResponse = [];

        //add below loop in Promise all

        await Promise.all(
            issues.map(async issue => {
                console.log(issue.node_id);
                if (issue.node_id.charAt(0) === "I") {
                    const issueRes = await handleIssueEvent({ issue }, octokit, repositoryOwner,
repositoryName);
                    const obj = { url : issue.url,title: issue.title, number: issue.number, username:
issue.user.login, responce: issueRes };
                    await issueResponse.push(obj);
                }
            }),
        );
        console.log(issueResponse);
        return res.status(200).json(issueResponse);
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: error.message });
    }
}

module.exports = { fetchIssues };
```

master/controller/project.js

```
                                        const Project = require('../models/project');
const User = require('../models/user');

const fetchProject = async (req, res) => {
    try {
        const project = await Project.findOne({ _id: req.params.projectId }).exec();
        res.status(200).json(project);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};
const fetchAllProject = async (req, res) => {
    try {
        const project = await Project.findOne({ }).exec();
        res.status(200).json(project);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};

const createProject = async (req, res) => {
    const project = new Project(req.body);
    try {
        await project.save();
        res.status(201).json(project);
    } catch (error) {
```

```
        res.status(400).json({ message: error.message });
    }
};

const getMyProjects = async (req, res) => {
    try {
        const user= await User.findOne({ email: req.params.email }).exec();
        if(!user){
            return res.status(404).json({ message: "User not found" });
        }
        const projects = await Project.find({ user: user._id }).exec();
        res.status(200).json(projects);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};

module.exports= { fetchProject, createProject,fetchAllProject,getMyProjects };
```

master/controller/pull_request.js

```
                                                const { Octokit } = require("@octokit/rest");
const config = require("../config/index");
const User = require("../models/user");
const Project = require("../models/project");
const askLLM = require("../service/askLLM");
const fetchCodeUtil = require("../controller/fetchCode");

const initGithub = async (req, res) => {
    const project = await Project.findOne({ _id: req.params.id }).exec();
    const owner = await User.findOne({ _id: project.user }).exec();

    const repositoryOwner = project.repository_owner || owner.github.username; // Replace with the
repository owner's username
    const repositoryName = project.repository_name; // Replace with the repository name
    const accessToken = project.access_token || process.env.token;
    console.log(repositoryOwner + " " + repositoryName); // Your GitHub Personal Access Token
    console.log(accessToken);

    const octokit = new Octokit({ baseUrl: "https://api.github.com", auth: accessToken });

    return { octokit, repositoryOwner, repositoryName };
};

async function handlePullRequestEvent(payload, octokit, repositoryOwner, repositoryName) {
    try {
        const prTitle = payload.pull_request.title;
        const prBody = payload.pull_request.body;

        // Fetch code changes from the pull request
        const { data: diff } = await octokit.rest.pulls.get({
            owner: repositoryOwner,
            repo: repositoryName,
            pull_number: payload.pull_request.number,
            mediaType: {
                format: "diff", // Get the diff format of the changes
```

```
            },
        });

        const reviewPrompt = `You are a PR reviewer A new pull request titled "${prTitle}" has been opened:\n\n${prBody}\n\nCode Changes:\n${diff}\n\nPlease review and provide feedback regarding any changes if required or whether it should be merged or not.`;
        console.log("Review Prompt:", reviewPrompt);

        const reviewResponse = await askLLM(reviewPrompt);
        console.log("LLM Review:", reviewResponse);

        // Add your logic to post the review comment back to GitHub using octokit
        // For example:
        await octokit.rest.pulls.createReview({
          owner: repositoryOwner,
          repo: repositoryName,
          pull_number: payload.pull_request.number,
          body: reviewResponse.response,
          event: "COMMENT", // or 'APPROVE', 'REQUEST_CHANGES', etc.
        });
        console.log("Review comment posted successfully");

        const response = reviewResponse.response;
        return { reviewPrompt, response };
      } catch (error) {
        console.error("Error:", error.message);
      }
    }

    // Example usage:
    // handlePullRequestEvent(payload);
    // Example: Listen for a new pull request event
    async function listenForPullRequests(req, res) {
      try {
        const { octokit, repositoryOwner, repositoryName } = await initGithub(req, res);
        const resp = await octokit.rest.pulls.list({
          owner: repositoryOwner,
          repo: repositoryName,
          state: "open",
        });

        let prs = [];
        // Process the response and trigger the appropriate function
        console.log("Pull Requests:", resp.data);

        await Promise.all(
          resp.data.map(async pr => {
            if (pr.node_id.charAt(0) === "P") {
              const { reviewPrompt, response } = await handlePullRequestEvent({ pull_request: pr }, octokit, repositoryOwner, repositoryName);
              console.log("Review Prompt:", reviewPrompt);
              console.log("Response:", response);
              await prs.push({ pr:reviewPrompt, review:response, number:pr.number, title:pr.title, body:pr.body, user:pr.user.login, url:pr.url});
            }
          }),
```

```
        );

        res.status(200).json({ message: "Pull Requests processed successfully", prs });
    } catch (error) {
        console.error("Error:", error.message);
        res.status(500).json({ message: error.message });
    }
}

module.exports = listenForPullRequests;
```

master/controller/status.js

```
                                        const axios = require("axios");
const askLLM = require("../service/askLLM");
const { fetchCodeUtil } = require("./fetchCode");
const prompts = require("../constants/prompts");
const Project = require("../models/project");
const mongoose = require("mongoose");

const framePrompt = async id => {
    try {
    //     console.log(id);
        const project = await Project.findOne({ _id: id }).exec();
        if (!project) {
            throw new Error(`Project with ID ${id} not found`);
        }

        let statusPrompt = prompts.statusPrompt || ''; // initialize to empty string if undefined
        // console.log(project);
        // console.log(statusPrompt);

        // append features json to statusPrompt string
        statusPrompt += JSON.stringify(project.features);
        return statusPrompt;
    } catch (error) {
        console.log(error);
        throw new Error(`Error framing prompt: ${error.message}`);
    }
};

const getStatus = async (req, res) => {
    try {
        const prompt = await framePrompt(req.params.projectId);
        // const outputPath = await fetchCodeUtil(req, res);
        const status = await askLLM(prompt);

        if (!status || !status.response || status.response === 'undefined') {
            throw new Error('Invalid response from askLLM');
        }
        console.log("-----------")
        console.log(status.response);

        const st = JSON.parse(status.response);
        if (!st) {
            throw new Error('Invalid JSON response');
```

```
        }

        st.date = new Date();

        const project = await Project.findOne({ _id: req.params.projectId }).exec();
        if (!project) {
            throw new Error(`Project with ID ${req.params.projectId} not found`);
        }

        project.statuses.push(st);
        await project.save();
        res.status(200).json(st);
    } catch (error) {
        console.log(error);
        res.status(500).json({ message: error.message });
    }
};


module.exports = { getStatus };
```

master/controller/user.js

```
                                    const User = require('../models/user');

const fetchUser = async (req, res) => {
    try {
        const user = await User.findOne({ _id: req.params.userId }).exec();
        res.status(200).json(user);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};
const fetchAllUsers = async (req, res) => {
    console.log("======")
    try {
        console.log("4567890")
        const users = await User.find({});
        res.status(200).json(users);
    } catch (error) {
        console.log("----------")
        res.status(500).json({ message: error.message });
    }
};



const createUser = async (req, res) => {
    try {
        console.log(req.body);
        const user = new User(req.body);
        console.log(user);
        await user.save();
        res.status(201).json(user);
    } catch (error) {
        res.status(400).json({ message: error.message });
```

```
    }
};

module.exports = { fetchUser, createUser ,fetchAllUsers};
```

master/models/chatbot.js

```
const mongoose = require('mongoose');

const chatbotSchema = new mongoose.Schema({
  query: {
    type: String,
    required: true,
  },
  response: {
    type: String,
    required: true,
  },
  date: {
    type: Date,
    default: Date.now,
  },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
  },
});

const Chatbot = mongoose.model('Chatbot', chatbotSchema);

module.exports = Chatbot;
```

master/models/document.js

```
const mongoose = require('mongoose');

const documentSchema = new mongoose.Schema({
  document_content: {
    type: String,
    required: true,
  },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
  },
});

const document = mongoose.model('document', documentSchema);

module.exports = document;
```

master/models/project.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const projectSchema = new Schema({
```

```
    name: String,
    description: String,
    repository_name: String,
    repository_url: String,
    repository_owner: String,
    access_token: String,
    features: [
        {
            name: String,
            checklist: [
                {
                    name: String,
                    status: Boolean,
                },
            ],
        },
    ],
    statuses: [
        {
            date: Date,
            features: [
                {
                    name: String,
                    status: String,
                    percentage: Number,
                    description: String,
                    checklist: [
                        {
                            name: String,
                            status: String,
                            percentage: Number,
                            description: String,
                        },
                    ],
                },
            ],
        },
    ],
    user: {
        type: Schema.Types.ObjectId,
        ref: "User",
    },
});

const Project = mongoose.model("Project", projectSchema);

module.exports= Project;
```

master/models/user.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const userSchema = new Schema({
    name: String,
    email: String,
```

```
    password: String,
    role: String,
    github: {
        id: String,
        token: String,
        username: String,
    },
});

const User = mongoose.model('User', userSchema);
module.exports= User
```

master/package.json

```
                                    {
  "name": "statusquo_backend",
  "version": "1.0.0",
  "description": "INC",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app.js"
  },
  "author": "The Error404 Team",
  "license": "ISC",
  "dependencies": {
    "@octokit/rest": "^20.0.2",
    "axios": "^1.6.8",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.18.3",
    "express-ws": "^5.0.2",
    "form-data": "^4.0.0",
    "mongoose": "^8.2.4",
    "node-fetch": "^3.3.2",
    "octokit": "^3.1.2",
    "pdfkit": "^0.14.0",
    "websocket": "^1.0.34"
  },
  "type": "commonjs"
}
```

master/requirements.txt

```
                                    ÿýÿýa  s À-  y Ð=  . @.  Ð

c 0e Àe  a @e Ð=  .  8 à0

i ðh @t  = Ð3 à9 à3

i ðs •g àa À= Ð1 à3 à1

n ào @a @e @- @y  e 0= Ð0 à6 à0

n •i ð= Ð4 à3 à0

s •n 0- @i Ðe ðu @= Ð4 à0 à3
```

t @r 0= Ð2 0. . Ð

e u @i `u Às ðu 4 Ð= @. 2 à3

s @= Ð0 à0 à2

0e t •f •= Ð2 2 @. . Ð

0h r 0e @- ào m l •z Pr Ð= 0. 0. Ð

0l •c °= Ð8 à1 à7

0o Ào a Ða Ð= . @. `Ð

@a @a 0l s 0e 0- s ðn Ð= . `. @Ð

@a @a 0e @s Ð= . 8 à0

@e r Pc t Pd Ð= . . 4

@i Àl Ð= . 0. €Ð

@i t •j 0o à= Ð1 à0 à8

@i 0t o Ð= . •. Ð

@m Ðt e P= Ð0 à1 à8

@o 0s @r •n p_ a s Pr Ð= . 6

Pt •l 0= Ð1 à7 à0

Px 0e t •o àg o Pp Ð= . . Ð

`a 0t p •= Ð0 à1 0 à0

`i Àe Ào 0k Ð= 0. 3 à3

`r ðz Pn Ài 0t Ð= . @. Ð

`s 0p Pc Ð= 0 4 à2 à0

po ðg Àe p •s Ðc ðm Ðo à- r ðt ðs Ð= . `3 à0

pr Pe àl Pt Ð= 0. . 0Ð

€1 = Ð0 à1 @. Ð

€t @p 0o e Ð= . . @Ð

€t @p €= Ð0 à2 p. Ð

€u pg •n pf c P- €u = Ð0 à2 . Ð

•d àa Ð= 0. `Ð

•m o t Ài _ e 0o Pr 0e 0= Ð6 à4 à0

i àj 2 Ð= 0. . 0Ð

o l •b Ð= . 0. Ð

Àl m - •n @e €= Ð0 à1 . 9

Àl m - •n @e €- g Pn @- ðp Pn i Ð= . . PÐ

Àl m - •n @e €- 0l •= Ð0 à1 à9

Àl m - •n @e €- 0o e Ð= . 0 à1 •Ð

Àl m - •n @e €- Pm e @d •n ps Ðo e àa •= Ð0 à1 à6

Àl m - •n @e €- •n @i 0e 0- Ða àa pe @- Àl m - 0l ðu @= Ð0 à1 à4

Àl m - •n @e €- Àe pa 0y Ð= . •. @8

Àl m - •n @e €- Àl Ðs Ðo e àa •= Ð0 à1 à9

Àl m - •n @e €- Ðu Àt •- Ðo @a À- Àl Ðs Ðo e àa •= Ð0 à1 à4

Àl m - •n @e €- r ðg a Ð- ðp Pn i Ð= . . @Ð

Àl m - •n @e €- u Ps @i ðn Ðg Pn Ðo e àa •= Ð0 à1 à3

Àl m - •n @e €- e d Pr 0- `i Àe Ð= . . 1

Àl m - •n @e €- e d Pr 0- Àl m - a s P= Ð0 à1 à3

Àl m - a s P= Ð0 à3 à9

Àl m i àd Px Ðp •- 0l •e àt Ð= . . 3

Ða k @o pn Ði @- y Ð= 0. . Ð

Ða k Pp 0a `e Ð= . . PÐ

Ða s €m l Ào p= Ð3 à2 . Ð

Ðd Pr À= Ð0 à1 à2

Ðp Ða @h Ð= . 0. Ð

Ðu Àt •d •c @= Ð6 à0 à5

Ðu Àt •p o 0e 0s Ð= . p0 à1 `Ð

Ðy y Ðe €t Pn 0i ðn 0= Ð1 à0 à0

àe 0t Ða 0y àc •o Ð= . `. Ð

àe @w ðr °x Ð= 0. . Ð

àl @k Ð= 0. €. Ð

àu Ðp •= Ð1 à2 `. @Ð

ðp Pn i Ð= . 6 à1

a 0k g •n p= Ð2 @. Ð

a àd s Ð= . . Ð

e `t Ð= . 0 à0

i Àl ðw Ð= 0 à2 à0

r ðm •s P= Ð2 à3

r ðt ðb Pf Ð= 0. 0 à3

s Pt •l Ð= P. •. €Ð

y r o p= Ð1 P. . Ð

y r o p- €o @f •x Ð= . `Ð

y @a àt •c Ð= . `. @Ð

y @a àt •c ðc ðr P= Ð2 à1 `. 0Ð

y pm Pn @s Ð= . 7 à2

y Ðu D `= Ð1 à2 0. 6

y Ðu D `b Ð= . 3 à2 Ð

y d `= Ð4 à1 à0

y @h ðn Ðd t Pu @i À= Ð2 à9 à0 àp ðs @0

y @h ðn Ðd ðt Pn `= Ð1 à0 à1

y @z Ð= 0 4 à1

y •A ÐL Ð= `. . Ð

e pe €= Ð2 2 0. 2 à2 PÐ

e u Ps @s Ð= . 01 à0

i 0h Ð= 3 à7 à1

0a `e @e às ðr 0= Ð0 à4 à2

0c •k •t Ðl Pa n Ð= . @. . o 0t Ð

0c •p •= Ð1 à1 . Ð

0h @a = Ð1 à7 à1

0i €= Ð1 à1 `. Ð

0n •f `i ð= Ð1 à3 à1

0o Pp 0i Pv P= Ð2 à5

0Q ÀA Àc €e Ðy Ð= . . 8

0t r Àe @t P= Ð0 à3 `. 0Ð

0t i r @f Ð= . . 6

0y Ðp •= Ð1 à1 Ð

@e àa 0i @y Ð= €. . 0Ð

@e às ðr `l ðw Ðd t s Pt 0= Ð4 à9 à4

@e às ðr `l ðw Ðm Pt d t = Ð1 à1 @. Ð

@e m 0o Ào = Ð2 à4 à0

@h e d o ðl 0t À= Ð3 à4 à0

@i °t ðk Pn Ð= . `. Ð

@o °e ài e s Ð= . 5 à2

@o Ðl Ð= . 0 à2

@o c €= Ð2 à2 à1

@q @m Ð= @. `6 à2

@r n 0f ðr Ðe s Ð= @. 09 à1

@r À= Ð0 à8 à1

@y i àg Ði às e 0t Ð= . •. Ð

@y i àg ðe €t Pn 0i ðn 0= Ð4 à1 . Ð

@y o Ð= . p. 0Ð

@z @a @a Ð= 0 4 à1

Pr Àl •b 0= Ð2 à2 à1

Pv •c ðr à= Ð0 à2 €. Ð

pr p @= Ð1 à1 `. Ð

€x €a 0h Ð= 0. @. Ð

•a l Ð= . •. @Ð

 i p Ð= 0. 8 à1

master/routes/chatbot.js

```
                                    const express = require('express');
const expressWs = require('express-ws');

const app = express();
expressWs(app);

const router = express.Router();
const chatbotController = require('../controller/chatbot');

router.ws('/chatbot-query', (ws, req) => {
  ws.on('message', async (message) => {
    try {
      console.log("qwertyuio")
      const data = JSON.parse(message);
      await chatbotController.chatbotQuery(ws, data);
    } catch (e) {
      ws.send(JSON.stringify({ message: 'Error processing query', error: e.message }));
    }
  });

  ws.on('close', () => {
    console.log('Connection closed for /chatbot-query');
  });
});

router.ws('/chatbot-response', (ws, req) => {
  ws.userId = req.query.userId;

  ws.on('message', async (message) => {
    const data = JSON.parse(message);
    await chatbotController.chatbotOutputResponse(ws, data);
  });

  ws.on('close', () => {
    console.log('Connection closed for /chatbot-response');
  });
});
router.ws('/create_doc', (ws, req) => {
  ws.userId = req.query.userId;

  ws.on('message', async (message) => {
    const data = JSON.parse(message);
    await chatbotController.saveToDocument(ws, data);
  });

  ws.on('close', () => {
    console.log('Connection closed for /chatbot-response');
```

```
  });
});

module.exports = router;
```

master/routes/code.js

```
const express = require('express');
const {fetchCommitWiseCode, fetchCurrentCode} = require('../controller/fetchCode');

const router = express.Router();

router.get('/current/:projectId', fetchCurrentCode);
router.get('/commit-wise/:projectId', fetchCommitWiseCode);

module.exports = router;
```

master/routes/document.js

```
const express = require("express");
const documentController = require("../controller/document");
const documentRouter = express.Router();

documentRouter.post("/add_to_document", documentController.addStringToDocument);
documentRouter.get("/genrate_document/:id", documentController.genrate_pdf);
module.exports = documentRouter;
```

master/routes/index.js

```
const userRouter = require('./user');
const projectRouter = require('./project');
const fetchCodeRouter = require('./code');
const statusRouter = require('./status');
const chatbotRouter = require('./chatbot');
const documentRouter = require('./document');
const pullRequestRouter = require('./pull_request');
const issueRouter = require('./issue');

module.exports = (app) => {
    app.use('/user', userRouter);
    app.use('/project', projectRouter);
    app.use('/status', statusRouter);
    app.use('/fetch-code', fetchCodeRouter);
    app.use('/chatbot', chatbotRouter);
    app.use('/document', documentRouter);
    app.use('/pull-request', pullRequestRouter);
    app.use('/issue', issueRouter);
}
```

master/routes/issue.js

```
const express = require("express");

const {fetchIssues} = require("../controller/issue");

const issueRouter = express.Router();

issueRouter.get("/suggest/:id", fetchIssues);
```

```
module.exports = issueRouter;
```

master/routes/project.js

```
const express = require('express');
const {fetchProject, createProject,fetchAllProject,getMyProjects} =require('../controller/project');

const router = express.Router();

router.get('/getAll', fetchAllProject);
router.get('/getMyProjects/:email', getMyProjects);
router.get('/:projectId', fetchProject);
router.post('/', createProject);

module.exports =router;
```

master/routes/pull_request.js

```
const express = require("express");
const listenForPullRequests = require("../controller/pull_request");

const pullRequestRouter = express.Router();

pullRequestRouter.get("/review/:id", listenForPullRequests);

module.exports = pullRequestRouter;
```

master/routes/status.js

```
const express = require('express');
const {getStatus} = require('../controller/status');

const router = express.Router();

router.get('/:projectId', getStatus);

module.exports = router;
```

master/routes/user.js

```
const express = require('express');
const {fetchUser, createUser,fetchAllUsers} = require('../controller/user');

const router = express.Router();
router.get('/getall', fetchAllUsers);
router.get('/:userId', fetchUser);
router.post('/', createUser);

module.exports = router;
```

master/service/askLLM.js

```
const axios = require('axios');

const askLLM = async (prompt) => {
   try{
      console.log("AskLLM")
      console.log(prompt);
```

```
        const response = await axios.get(`http://localhost:6000/query/?query=${prompt}`);
        console.log(response.data)
        return response.data;
    }catch(error){
        console.log(error.message);
        return error.message;
    }
}

module.exports= askLLM;
```

master/service/fetchCodeServices.js

```
                                                    const fetchCommitList =
async(octokit,repositoryOwner, repositoryName)=> {
    try {
        const response = await octokit.request("GET /repos/{owner}/{repo}/commits", {
            owner: repositoryOwner,
            repo: repositoryName,
        });
        //console.log(response.data)
        return response.data;
    } catch (error) {
        console.error("Error fetching commit list:", error);
        throw error;
    }
}

async function fetchTree(octokit,commitSha,repositoryOwner, repositoryName) {
    try {
        const response = await octokit.request("GET /repos/{owner}/{repo}/git/trees/{tree_sha}", {
            owner: repositoryOwner,
            repo: repositoryName,
            tree_sha: commitSha,
            recursive: 1,
        });
        //console.log(response.data.tree)
        return response.data.tree;
    } catch (error) {
        console.error("Error fetching tree:", error);
        throw error;
    }
}

module.exports= {
    fetchCommitList,
    fetchTree,
};
```

master/service/upload.js

```
                                    const FormData = require("form-data");
const axios = require("axios");
const fs = require("fs");
const PDFDocument = require("pdfkit");

async function uploadFile(filePath) {
```

```
    //replace \ with / in the file path
    filePath = filePath.replace(/\\/g, "/");
    console.log("Uploading file:", filePath);
    const formData = new FormData();
    formData.append("file", fs.createReadStream(filePath));

    const url = "http://localhost:6000/uploadfile/";

    try {
        const response = await axios.post(url, formData, {
            headers: {
                ...formData.getHeaders(),
                "Content-Type": "multipart/form-data",
            },
        });
        console.log(response.data);
    } catch (error) {
        console.error("Error uploading file:", error.message);
    }
}

//uploadFile("D:/cs programs/ProjectAI/master/StatusQuo-backend_20240406_latest.pdf")

module.exports = {uploadFile};
```

readme.md

```
# statusQuo Backend

This repository contains the backend code for the statusQuo project. The backend is built using Node.js and FastAPI, and it consists of various components including configuration files, controllers, middleware, models, routes, utilities, and services.

## Node.js Backend

The Node.js backend is located in the `node_backend` directory. It contains the following components:

- **Config**: This directory contains configuration files for the backend, such as database connection settings, API keys, and environment variables.

- **Controller**: The controller directory contains the logic for handling incoming requests and generating responses. It includes functions for fetching data from the GitHub repository.

- **Middleware**: Middleware functions are used to intercept and modify incoming requests before they reach the controller. This directory contains middleware functions for authentication, error handling, and more.

- **Model**: The model directory contains the data models used by the backend. These models define the structure and behavior of the data stored in the database.

- **Routes**: The routes directory contains the API routes for the backend. Each route maps to a specific URL endpoint and is associated with a controller function.

- **Utils**: The utils directory contains utility functions that are used throughout the backend codebase. These functions provide common functionality such as data validation, formatting, and error handling.
```

- **Services**: The services directory contains additional services used by the backend. This includes the `llm_service` which is responsible for calling the LLM (Language Model) with a prompt PDF and returning the answer.

FastAPI Backend

The FastAPI backend is located in the `fastapi_backend` directory. It contains the `llm_service` which is responsible for calling the LLM with a prompt PDF and returning the answer.

Getting Started

To get started with the statusQuo backend, follow these steps:

1. Clone the repository to your local machine.
2. Install the required dependencies by running `npm install` in the `node_backend` directory and `pip install -r requirements.txt` in the `fastapi_backend` directory.
3. Configure the backend by updating the necessary configuration files in the `config` directory.
4. Start the Node.js backend by running `npm start` in the `node_backend` directory.
5. Start the FastAPI backend by running `uvicorn main:app --reload` in the `fastapi_backend` directory.

That's it! You should now have the statusQuo backend up and running.