LLM-Agile-2

Company:
Argo

Project Managers:
Mark Bentsen
Gleb Radchenko

Backend Developers:
Asa Mcdaniel
Team Lead, PrivateGPT Integration
Daniel Morandi
System Architect, API Dev, Firebase
Ubadah Saleh
Jira and Online LLM Integration
Guadi Morado
Jira Integration, Basic Security

Frontend Developers
Faizul Anis
UI, All Database Integration
Rishi Meka
UI, LLM Communication & Jira Integration

Table of Content

Notes:

- There are three separate processes that can be in their own 3 files.
- The github https://github.com/RailgunDotEnc/LLM-Agile-2 has the developer format for testing on one computer. Simply fill in the empty files with their appropriate matching githubs.
- Empty folders are a reference if you download privategpt and Argo-MultiLLM-UI individually instead of using https://github.com/RailgunDotEnc/LLM-Agile-2
- **Backend**:
  - All these files are expected to run on a local server on a separate device of the client
  - This is composed of two sections, there server which is referred to as API.py and the local LLM called privateGPT.
  - API.py and privateGPT are not codependent and can run separately
  - All interactions with the exception of adding files to privateGPT are done in API.py.
  - PrivateGPT runs on local host
  - API.py connects to both PrivateGPT and is called by the client
    For the following actions
    - Jira communication
    - Firebase Communication
    - Client Communication
    - Online LLM calls
    - Local LLM calls
- **FrontEnd**:
  - This is meant to run on the clients personal computer
  - Does not do any heavy processing as all task are done in server
  - Simply an interaction point to the backend
  - Does not have an admin mode for direct access to databases

Backend Manual Setup

Download from GitHub or Zip (Sever)
https://github.com/RailgunDotEnc/LLM-Agile-2 (All files)
Download from GitHub or Zip (Private gpt): not our original code
https://github.com/zylon-ai/private-gpt (Private gpt only)

Included in downloads	Not included
API.py	Settings.py
FireDB.py	Credentials.json
Jira.py	
Models.py	
Setup.py	
pipinstall.txt	

Pre Setup Download Requirements

Main API:

-Python 12.0 for API.py (Python 9.0-12.0 without PrivateGPT)
-Windows/Linux, Mac
-pipinstall.txt

Private Gpt
-Anaconda with Python 11.0 for Private GPT
-Windows (Mac and Linux support for everything except private gpt)
-Nvidia drives
-https://docs.chocolatey.org/en-us/choco/setup

Setup Main API

1) Pip install all files inside \Backend\pipinstall.txt
2) Create file Settings.py in \Backend

```
—----------------------------------------------
#API Key stuff
default_key="xyz"
key_levels=["abcde","bcdef","cdefg","defgh","efghi"]
#authenticate to firebase
cred_file="credentials.json"
database_url="<http>"
# Model API keys
GOOGLE_API_KEY="<key>"
OPENAI_API_KEY = "<key>"
CLUADE_API_KEY = "<key>"
#Jira
#included in API.py
jira_domain = "<http>"
#Server Settings
Port=<int>
#PrivateGPT use documents
context=True
—----------------------------------------------
```

3) Create google firebase https://firebase.google.com/.
    a) Make real real-time database
    b) Copy url https://<unique>.firebaseio.com/ from real-time database table
    c) Go to “project settings” and “Service accounts”
    d) Choose “Python” and click “Generate new private key”
    e) Save file in /Backend as “Credentials.json”

4) Run in Backend/ with “python API.py”
    a) If there is a start error restart it once (Error does not appear in linux)
    b) Windows may have issues ending the terminal and needs the terminal to be killed
    c) These errors are from “uvicorn” and can be ran manually instead https://fastapi.tiangolo.com/deployment/manually/.
    d) Linux needs to use sudo python API.py

Setup Private GPT

privateGPT is an open source tool that we used to satisfy the ‘query local documents’ requirement of the app. The following link is the documentation provided, though in our experience it misses a few steps.
https://docs.privategpt.dev/installation/getting-started/installation

1) Clone the Github Repository and navigate your cmd to it
   > git clone https://github.com/zylon-ai/private-gpt
   > cd private-gpt

2) Make sure you're using Python 3.11 for this application. The app uses poetry, a package handling tool, which as of the creation of this document requires (Python v. >= 3.11, < 3.12) to function.
   - This can be done through whichever form of version control you'd like to use.
3) Install poetry
   > pip install poetry

4) Install make
   > choco install make

5) Use poetry to install required packages and setup the app.
   > poetry install --extras "ui llms-llama-cpp embeddings-huggingface vector-stores-qdrant"
   > poetry run python scripts/setup
   *Make sure poetry is using 3.11. This is the part that requires python 3.11 be used.

6) Run the following commands in Windows cmd to start privateGPT. If you are using a different command line, look at the documentation to figure out the needed method.
   > cd …/PrivateGPT

   Documentation Method
   > set PGPT_PROFILES=local
   > make run

   Method I use in Backend/PrivateGPT
   > poetry run python -m uvicorn private_gpt.main:app --reload --port 8001

This is how to run privateGPT on your computer. Our application will use an API to listen to privateGPT url and functions as though it was a user in the basic application.

Backend Breakdown

Adding more endpoints for API in API.py

API.py

-Edit anything with <>

```
#Remove format using title
#HTTP extension
@app.get("/api/<endPoint>")
async def bard_call(request: Request,api_key: str,<Needed inputs>: str):
    #Checks if api key came from website
```

```
    if check_Key(api_key)!=-2:
        #Do process in another file from imports
        message=<File>.<function>()
        #Return json to be sent to client
        return message
    else:
        return {"message":"Access Error"}
```

Adding more models in Models.py

Model.py

-Edit anything with <>

-The first function is the model API calls which takes in a prompt string and returns a response string.

```
# Model
def <model>(prompt):
    <Model Logic>
)
    #Return string
    return f"{<return response string}"

def <model chat>(prompt,model_history):
    #Unpacking history form API for set blank if empty
    message,new_model_history,key_num,model_history=unpack_history(model_history)
    #Reformat prompt with prompt history
    new_prompt=prompt_reformatting(model_history,prompt)
    #Request from Claude
    response={"response":<model>(new_prompt)}
    #Reformat response to send back
    message=repack_history(new_model_history,model_history,key_num,prompt,response,message)
    return message
```

Model.py

-Edit anything with <>

-This is the first function called from API.py. Only needs to add the function call and the string that is expected from client when they put “&Model=<model name>”

```
#Call from API
client = OpenAI(api_key = ST.OPENAI_API_KEY)
def model_manager(Model_name,prompt,model_history):
```

```
    elif Model_name=="<model name> (called from front end client, no API.py changes
needed)":
        Result=<model>_chat(prompt, model_history)
    else:
        Result={"Error":"Model not found"}
    return Result
```

Updating or Modifying Security APY.py & Security.py

In APY.py
-Edit anything with <>
-Grab the website key. First three chars
-Skip middle 5 values which would be unique per user
-Grab last 5 chars which indicate security level [1-#] from array order
-Returns -2 if there is no api key. Made for ping call

```
#API Key Manager
#Example key: xyz12345abcde
def check_Key(key):
      #Grab the website key. First three chars
    print(key[0:3],ST.default_key)
      #Skip middle 5 values which would be unique per user

      #Grab last 5 chars which indicate security level [1-#] from array      order
    print(key[8:13],ST.key_levels[0])
      #If not from website and no key sent for ping test
    if not (key[0:3]==ST.default_key):
        return -2
    elif key[8:13]==ST.key_levels[0]:
        return 1
    elif key[8:13]==ST.key_levels[1]:
        return 2
    elif key[8:13]==ST.key_levels[2]:
        return 3
    elif key[8:13]==ST.key_levels[3]:
        return 4
    elif key[8:13]==ST.key_levels[4]:
        return 5
    else:
        return -1
```

Setting.py

-Edit anything with <>

-Default key is from the website itself that should be able to change in settings file

-key levels, can be changed to any 5 values unless API.py security is also changed

```
#API Key stuff
default_key="xyz"
key_levels=["abcde","bcdef","cdefg","defgh","efghi"]
```

In depth on how api calls look like and how data is structured

-This can also be found in /Backend/ReadMe.py

1) Security

```
import json
#How API keys work
webkey="xyz"
userkey="12345abcde"
apikey=webkey+userkey
#apikey=xyz12345abcde
#The key is split into currently 3 parts
#Part 1 xyz: This just makes sure that there is company access and is a legal key. This key will be added automatically by the website
#Part 2 12345: This is going to be a unique number based on user. If we have time and are done with everything else, we can add a verificaiton that the user exist.
#Part 3 abcde: This has 5 versions from abcde, bcdef, ...., This indicates the security level and the filtration of the dataset

#Add this to Settings.py
default_key="xyz"
key_levels=["abcde","bcdef","cdefg","defgh","efghi"]

#How to add Prompt to database
##############################################################################################--
```

2) Basic call to ping and check if server is up

```
#IP Addresses and ports#########################################################################################
###
```

```
ip="70.153.136.26"
port="333"
###This calls ping for server testing with no api key
htpp=f"http://{ip}:{port}"
###This calls ping for server testing with api key
http=f"http://{ip}:{port}/?api_key={apikey}"

##########################################################################################
```

3) Calls for Firebase manipulation

```
#Variable
Pusername="Daniel Morandi"
Ptitle="Apple Test Question"
Pprompt="What color is an apple"
PSLC=-1
Psecurity=-1


jsonPromptExample={"Ptitle":    #Title header
                    {"Pusername":{
                        "Prompt":"What color is an apple",
                        "models":{
                                    "gemini":
                                    {"model_history":{
                                        "Question_1":["What color is an apple","red"],
                                            "Question_2":["What was the first question","What
color is an apple"]}
                                    },
                                    "gpt":
                                        {"model_history":{
                                            "Question_1":["What color is an apple","red"],
                                                        "Question_2":["What was the first
question","What color is an apple"]}
                                        },
                                    "claude" :
                                     {"model_history":{
                                        "Question_1":["What color is an apple","red"],
                                            "Question_2":["What was the first question","What
color is an apple"]}
                                    }
```

```
                        },
                        "SDLC": "PSLC",
                        "Security": "Psecurity"
                        }
                    }
                }


#Http call to add prompt data
http=f"http://{ip}:{port}/api/AddPrompt?api_key={apikey}&Data={jsonPromptExample}"

#How                          to                              add                          Format
#############################################################################################
Fusername="Daniel Morandi"
Title="This is a title"
Fformat="In the structure of a table"
jsonFormatExample={
                Title:{
                    Fusername:Fformat
                    }
                }
#Http call
http=f"http://{ip}:{port}/api/AddFormat?api_key={apikey}&Data={jsonFormatExample}"


#Rest                         of                          https                          request
#############################################################################################
#Find all prompts: only returns titles
http=f"http://{ip}:{port}/api/ScanAllPrompts?api_key={apikey}"
#Find all formats: only returns titles
http=f"http://{ip}:{port}/api/ScanAllFormats?api_key={apikey}"

#Find all prompts for title based on security level
PromptTitle="Apple Test Question"
http=f"http://{ip}:{port}/api/FindPrompt?api_key={apikey}&Title={PromptTitle}"

#Find All formats for title
FormatTitle="This is a title"
http=f"http://{ip}:{port}/api/FindFormat?api_key={apikey}&Title={FormatTitle}"
```

```
#Remove Prompt Title based on ur own name
Username="Daniel Morandi"
Title="Apple Test Question"
http=f"http://{ip}:{port}/api/RemovePrompt?api_key={apikey}&Username={Username}&Title={Title
}"

#Remove Format Title based on ur own name
Username="Daniel Morandi"
Title="This is a title"
http=f"http://{ip}:{port}/api/RemoveFormat?api_key={apikey}&Username={Username}&Title={Title
}"
```

4) How to call models with and without history

```
#How                    to                    Request                    from                    model
#############################################################################################

#Without History
model="gpt"
prompt="What color is an apple"
http=f"http://{ip}:{port}/api/Model?api_key={apikey}&Model={model}&Prompt={prompt}"

#With History
model="gpt"
prompt="What color is an apple"
history={"model_history":{"Question_1":["What color is an apple","red"],
"Question_2":["What was the first question","What color is an apple"]}
                                        }
http=f"http://{ip}:{port}/api/Model?api_key={apikey}&Model={model}&Prompt={prompt}&History={
history}"
```

5) Request to get_story and get_story_list for Jira. Only get_story_list is currently implemented

```
#Hot                    to                    Request                    from                    Jira
#############################################################################################
#Get these from Jira
issue_key="FP-1"
email=""
api_token=""
```

```
http=f"http://{ip}:{port}/api/get_story?api_key={apikey}&issue_key={issue_key}&email={email}
&api_token={api_token}"

project_key="FP"
email=""
api_token=""
http=f"http://{ip}:{port}/api/get_story_list?api_key={apikey}&project_key={project_key}&emai
l={email}&api_token={api_token}"
```

6) Setting again

```
#Files you need to make for api to work
#################################################################################
#Settings.py
#-----------------------------
#Security
default_key="xyz"
key_levels=["abcde","bcdef","cdefg","defgh","efghi"]

#authenticate to firebase
cred_file=""
database_url=""

# Setup API keys
GOOGLE_API_KEY=""
OPENAI_API_KEY = ""
CLUADE_API_KEY = ""

#Port
Port="364"
#---------------------------
#credentils.json
#Get from firebase. Ask guidi
```

PrivateGPT Adding files

This can be performed through the baseline privateGPT app by clicking the ‘Upload File’ button and choosing which files for it to ingest. Files will stay on the computer they are ingested on and become the knowledge base of that instance of privateGPT. To allow client users to add documents, there is a prebuilt API call in API.py. This function takes any amount of plaintext and ingests it in the same way as you would in the browser. It is not currently callable from the front-end, but it would just need a button and an API call built there.

PrivateGPT Replacing LLM

When installing privateGPT, you can modify the settings-local.yaml file in the privateGPT folder to download which local model you want to use.

FrontEnd Manual Setup

Download from github or Zip

https://github.com/RailgunDotEnc/LLM-Agile-2 (This contains all Files)
https://github.com/rishimeka/Argo-MultiLLM-UI (Frontend only)

Included in downloads	Not included
.idea	.env.local
public	npm install files
src	
README.md	
next.config.mjs	
package-lock.json	
package.json	
tsconfig.json	

Pre Setup Download Requirements

-npm
-windows/mac/linux

Setup Client UI

1) Download from github or Zip
2) In 'Frontend/' create .env.local with the following using https://auth0.com/ & https://youtu.be/0b8qHQlc-hQ?si=Jy4Wg7fd5iHUGp6D

```
AUTH0_SECRET=''
AUTH0_BASE_URL='http://localhost:3000'
AUTH0_ISSUER_BASE_URL='https://<>.us.auth0.com'
AUTH0_CLIENT_ID=''
AUTH0_CLIENT_SECRET=''
```

3) run 'npm install'
4) Fix usersettings in 'Frontend/src/app/page.tsx with jiraAPIkey, BaseURL which needs ip and port. Updating websitekey is optional. More info below on How to 'Modify UI: User settings'
5) Run file with 'npm run dev'

How to Modify UI

User settings

Frontend/src/app/page.tsx

```
const [loading, setLoading] = useState(false);
    const websiteKey = 'xyz' // TODO: Add the process.env read t
    const jiraAPIkey = ''
    const [userKey, setUserKey] = useState<string>("12345abcde");
    const [PromptTitle, setPromptTitle] = useState<string>("");
    const [FormatTitle, setFormatTitle] = useState<string>("");
    const BaseURL = 'http://<ip>:<port>/api/';
```

- websitkey: guarantees that the api calls are from the website, and not from random users
- jiraAPIkey: receive from argo
- userkey: this is a default for user 12345 with clearance abcde which is basic 1 clearance
- BaseURL: needs ip and port from server

Current Acceptable models

```
const models = [
        {
            label: "Chat GPT",
            value: 'gpt'
        },
        {
```

```
            label: "Google Gemini",
            value: 'gemini'
        },
        {
            label: "Claude",
            value: 'claude'
        },
        {
            label: "Llama",
            value: 'llama'
        },
    ]
```

-Models that are currently available
-If models are added in backend, this needs to be updated

UI Demo

https://drive.google.com/file/d/19QjF-VycceROrKFdjJt5fDRefO3MImb8/view?usp=drive_link