Avatar AI

A. TEXT MESSAGE AVATAR AI

VR Chat World Creation (Unity)

Prior to world creation make sure, the following are downloaded and installed:
A. VRChat Creator Companion. (Download link Creator Companion from official website)
B. Unity Hub: (Download and install the version recommended by VRChat Unity ie; 2022.3.6f1. As of 2023 may
5. Setup the account for Unity Hub, so that the unity can be used. (Sign up or login if you already have the account)
D. Once the installation and login is completed, Open Creator Creator and create a World project. This open ups the unity. Please make sure the unity is opened with necessary components like VRChatSDK.
E. Once, this is completed follow the below instructions to create a world for VRChat that has a text chat with AI facility.

1. Creating a World having text chat:

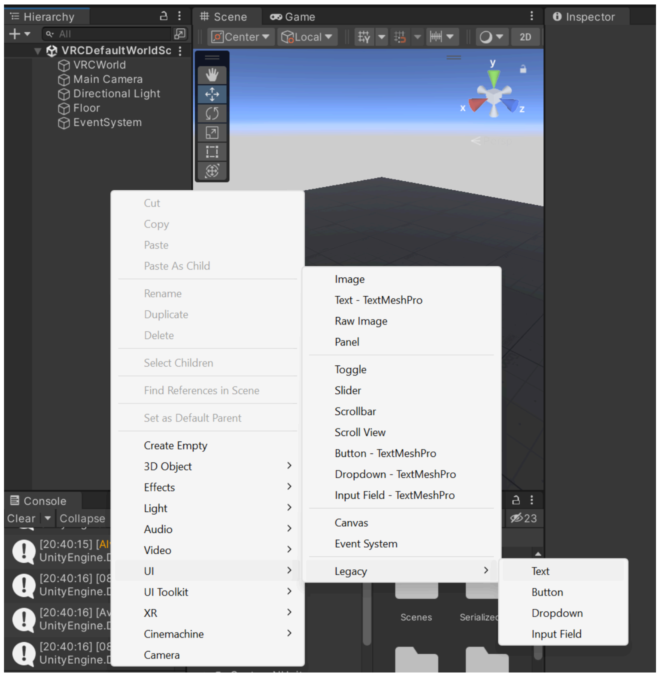
a. To create a world with chat UI we require 3 fundamental objects in our scene:

i. **Text (for output)** : For displaying the response from AI
ii. **Cube (act as button)** : For sending the question to FastAPI (AI code) and get the response back.
iii. **InputField** : For taking user question

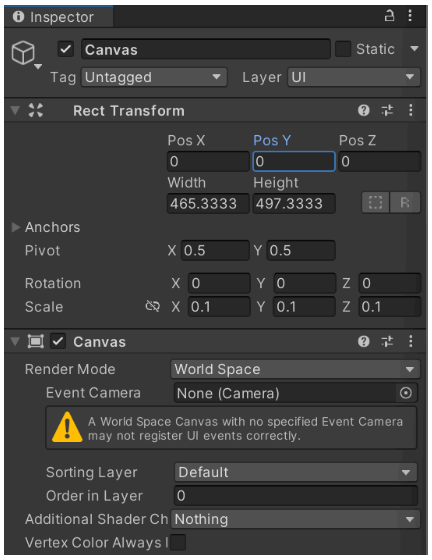
2. Steps for the creation of a chatting UI.

i. Text (for output)

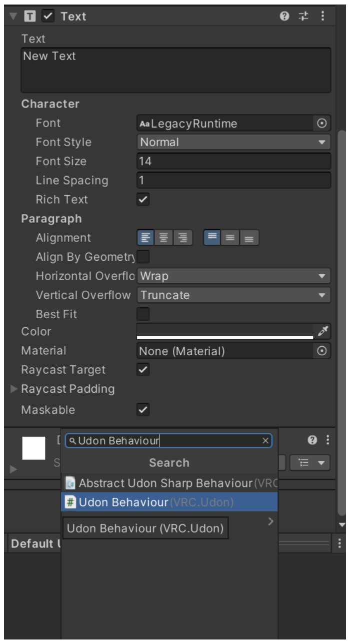
1. Step 1: In the **Hierarchy** do right click and add Text(Legacy) ie:
UI>Legacy>Text
Note: TextmeshPro can also be used instead of legacy text. I switched to Textmeshpro later in the project for this task, but to keep it simple lets use Text(legacy).



2. Step 2: Rename the “Text (Legacy)” in the Hierarchy to “ChatOutput”.
3. Step 3: Resize the Canvas and ChatOutput field for better view:
    a. Click Canvas which opens the “Inspector” for Canvas, then on its Render Mode select “World Space”
    b. This will let the resize and relocate Canvas using “Recet Transform”. Downscale the Scale of X, Y and Z all to 0.1


Inspector
Canvas — Static
Tag: Untagged — Layer: UI
Rect Transform
Pos X: 0 — Pos Y: 0 — Pos Z: 0
Width: 465.3333 — Height: 497.3333
Anchors
Pivot: X 0.5 Y 0.5
Rotation: X 0 Y 0 Z 0
Scale: X 0.1 Y 0.1 Z 0.1
Canvas
Render Mode: World Space
Event Camera: None (Camera)
A World Space Canvas with no specified Event Camera may not register UI events correctly.
Sorting Layer: Default
Order in Layer: 0
Additional Shader Ch: Nothing
Vertex Color Always I

4. Step 4: Add UdonBehaviour on ChatOutput.
    a. Go to Inspector window of ChatOutput
    b. Click Add Component and search Udon Behaviour and add it.


Text

Text: New Text

Character
- Font: LegacyRuntime
- Font Style: Normal
- Font Size: 14
- Line Spacing: 1
- Rich Text: ✓

Paragraph
- Alignment
- Align By Geometry
- Horizontal Overflow: Wrap
- Vertical Overflow: Truncate
- Best Fit

Color

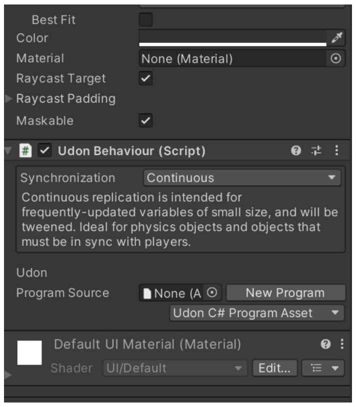
Material: None (Material)

Raycast Target: ✓

Raycast Padding

Maskable: ✓

Udon Behaviour
Search
Abstract Udon Sharp Behaviour (VRC
Udon Behaviour (VRC.Udon)
Udon Behaviour (VRC.Udon)
Default


Best Fit

Color

Material: None (Material)

Raycast Target: ✓

Raycast Padding

Maskable: ✓

Udon Behaviour (Script)

Synchronization: Continuous

Continuous replication is intended for frequently-updated variables of small size, and will be tweened. Ideal for physics objects and objects that must be in sync with players.

Udon

Program Source: None (A | New Program

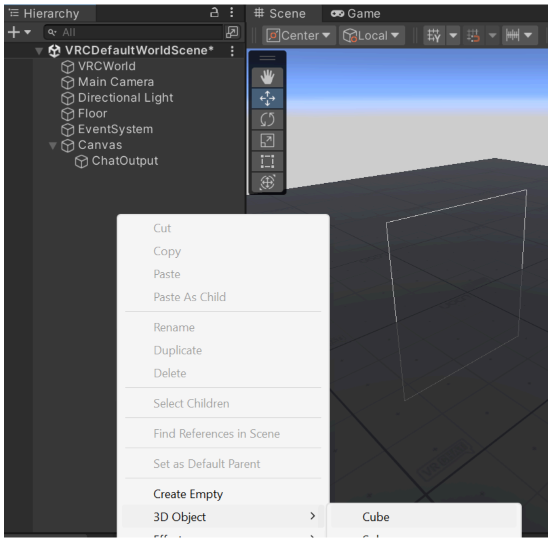
Udon C# Program Asset

Default UI Material (Material)

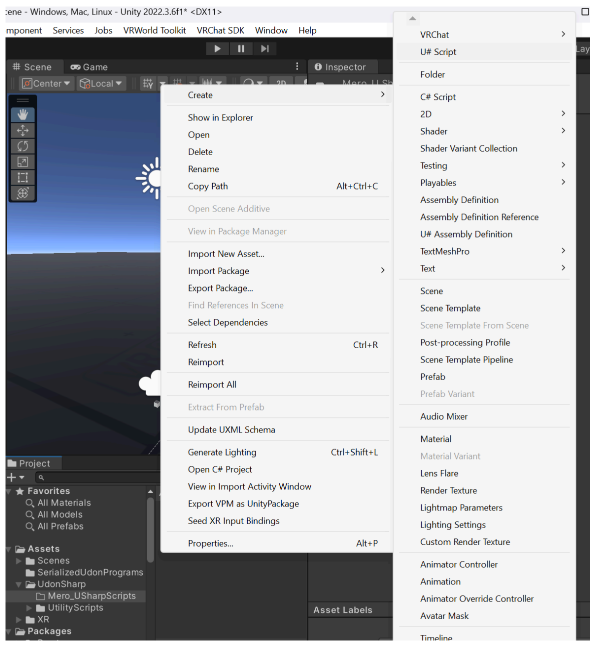
Shader: UI/Default | Edit...

ii. Cube (Act as button)

1. Step 1: Create 3D Cube from Hierarchy window 3D Objects.



2. Step 2: Rename it as ChatButtonCube. Also rescale it to your requirement using Inspector window.
3. Step 3: On the Layer option in Inspector window keep the Layer as Default not UI.
4. Step 3: Adding a UdonSharp (U# Script) code to it.
    a. Create a U# script from Project window:

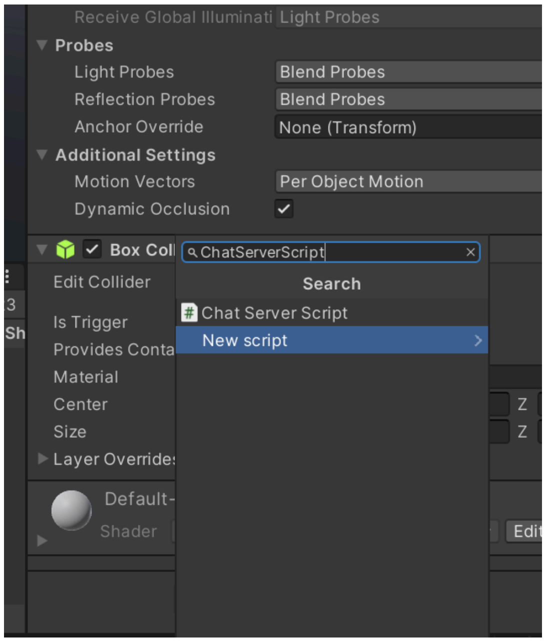


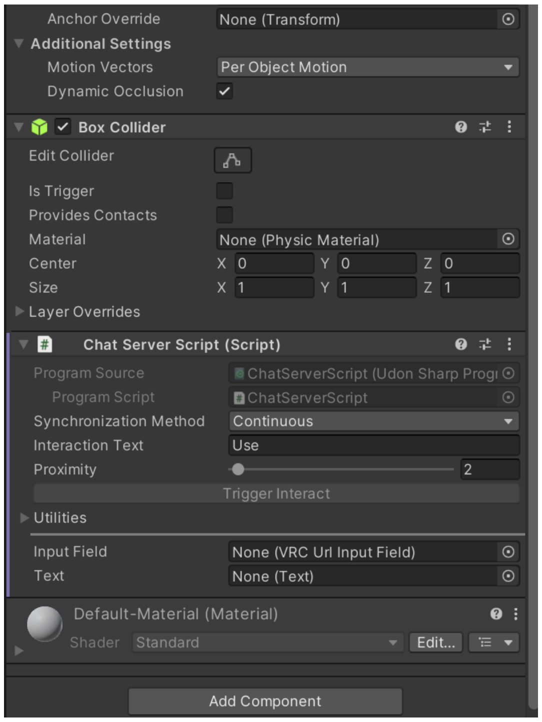
b. Rename the just created empty U# file as ChatServerScript then paste the following code: (Note: here the class name and file name should be same)

```
using UdonSharp;
using UnityEngine;
using VRC.SDKBase;
using VRC.Udon;
using UnityEngine.UI;
using VRC.SDK3.StringLoading;
using VRC.SDK3.Components;

public class ChatServerScript :
UdonSharpBehaviour
{
    [SerializeField] public VRCUrlInputField
_inputField;
    [SerializeField] private Text _text;
    private VRCUrl inputUrl;

    public override void Interact()
    {
        inputUrl = _inputField.GetUrl();

        VRCStringDownloader.LoadUrl(inputUrl,
this.GetComponent<UdonBehaviour>());
```

```
        _text.text = "Loading...";
        Debug.Log("URL : " + inputUrl);

    }
    public override void
OnStringLoadSuccess(IVRCStringDownload
result)
    {
        _text.text = result.Result;
    }
    public override void
OnStringLoadError(IVRCStringDownload result)
    {
        _text.text = "Error.";
        Debug.Log(result.Error);
    }
}
```

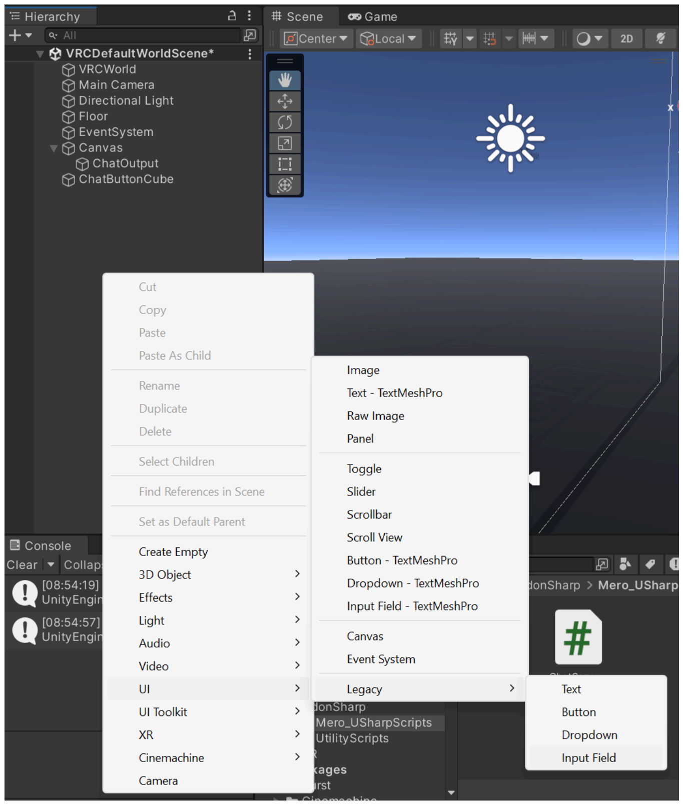
c. On the Inspector window of ChatButtonCube, add this U# script as a component. To do so, click Add Component and search ChatServerScript and click that and add.


Receive Global Illuminati Light Probes
Probes
Light Probes: Blend Probes
Reflection Probes: Blend Probes
Anchor Override: None (Transform)
Additional Settings
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓
Box Coll ChatServerScript
Search
Chat Server Script
New script

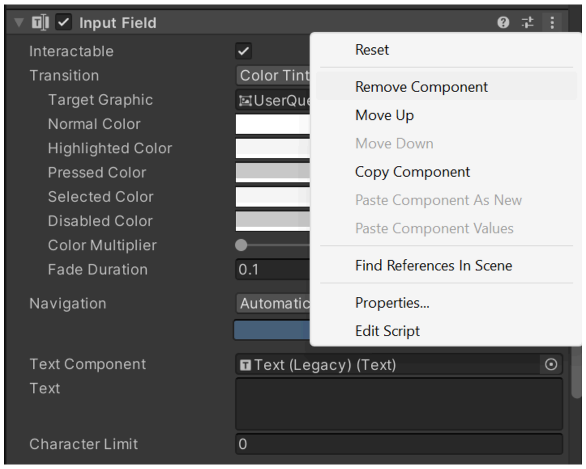

Anchor Override: None (Transform)
Additional Settings
Motion Vectors: Per Object Motion
Dynamic Occlusion: ✓
Box Collider
Edit Collider
Is Trigger
Provides Contacts
Material: None (Physic Material)
Center: X 0 Y 0 Z 0
Size: X 1 Y 1 Z 1
Layer Overrides
Chat Server Script (Script)
Program Source: ChatServerScript (Udon Sharp Progr
Program Script: ChatServerScript
Synchronization Method: Continuous
Interaction Text: Use
Proximity: 2
Trigger Interact
Utilities
Input Field: None (VRC Url Input Field)
Text: None (Text)
Default-Material (Material)
Shader: Standard
Add Component

iii. InputField:

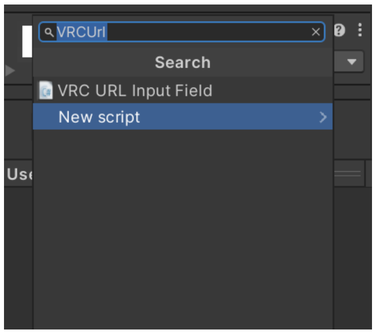
1. Step 1: To create InputField for user, right click in the Hierarchy window and add an Input Filed UI. ie: UI>Legacy>Input Field.



2. Rename this created “ InputFiled (Legacy)” as “UserQuestion”.Then resize it as per need.
3. Open the Inspector of this UserQuesiton object and make following changes:
    a. Change the Layer from “UI” to “Default”.
    b. Add “VRC UI Shape (Script) by clicking the “Add Component” and searching it. *This part is necessary else, the object will not let us input text in the field when Creator Companion is used.*
    c. Remove the Component called “InputField” of this UserQuestion and add a component called “VRCURLInputField”. You can search it after clicking “Add component” and add it.*(We need to remove the default input field inorder to use the VRCurlInputField)*



Interactable ✓
Transition: Color Tint
Target Graphic: UserQue
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier
Fade Duration: 0.1
Navigation: Automatic
Text Component: Text (Legacy) (Text)
Text
Character Limit: 0



d. After adding the VRCUrlInputField go to “Text Component” and where there is None(Text), change that to Text (Legacy) by either selecting from “Select Text window” that popups when you click the field or by dragging and releasing here from Hierarchy (UserQuestion>Text(Legacy)). And do the same for Placeholder as well.

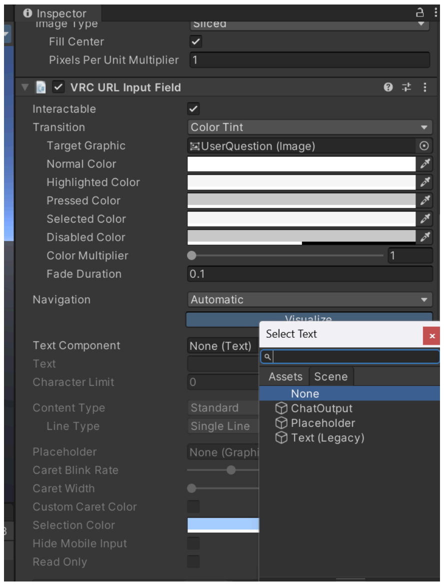
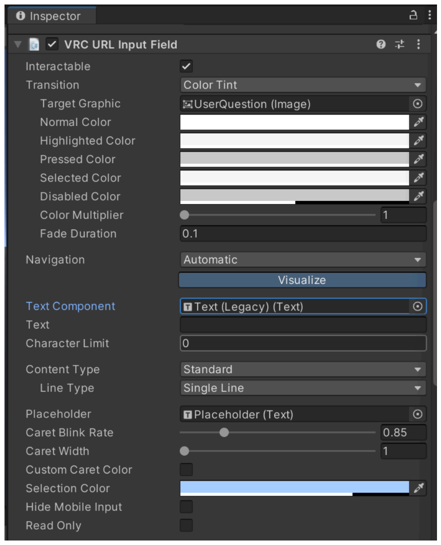


Image Type: Sliced
Fill Center ✓
Pixels Per Unit Multiplier: 1
Interactable ✓
Transition: Color Tint
Target Graphic: UserQuestion (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
Text Component: None (Text)
Text
Character Limit: 0
Content Type: Standard
Line Type: Single Line
Placeholder: None (Graphi
Caret Blink Rate
Caret Width
Custom Caret Color
Selection Color
Hide Mobile Input
Read Only



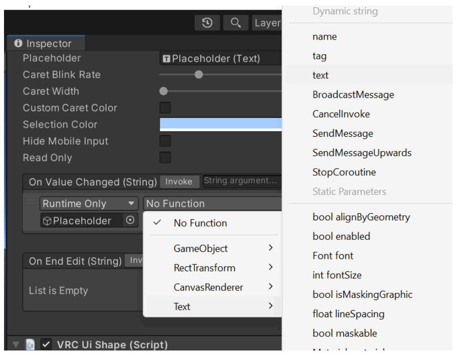
Inspector

VRC URL Input Field

Interactable	✓
Transition	Color Tint
Target Graphic	UserQuestion (Image)
Normal Color	
Highlighted Color	
Pressed Color	
Selected Color	
Disabled Color	
Color Multiplier	1
Fade Duration	0.1
Navigation	Automatic
	Visualize
Text Component	Text (Legacy) (Text)
Text	
Character Limit	0
Content Type	Standard
Line Type	Single Line
Placeholder	Placeholder (Text)
Caret Blink Rate	0.85
Caret Width	1
Custom Caret Color	
Selection Color	
Hide Mobile Input	
Read Only	

e. Add the “Placeholder” to “On Value Changed (String)” and “UserQuestion” to the “On End Edit (String)”. To add these:

i. For OnValueChanged: Click the “+” sign and drag the Placeholder (inside the UserQuestion in hierarchy) to the box below Runtime. Then, Choose the Text>text in its function.



Inspector

Placeholder	Placeholder (Text)
Caret Blink Rate	
Caret Width	
Custom Caret Color	
Selection Color	
Hide Mobile Input	
Read Only	

On Value Changed (String)

Runtime Only — No Function

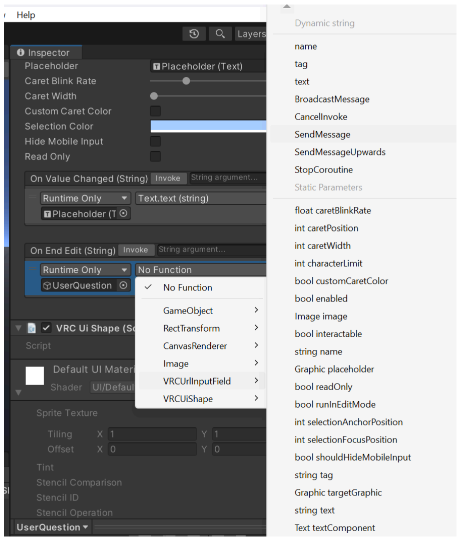
Placeholder

On End Edit (String)

List is Empty

VRC Ui Shape (Script)

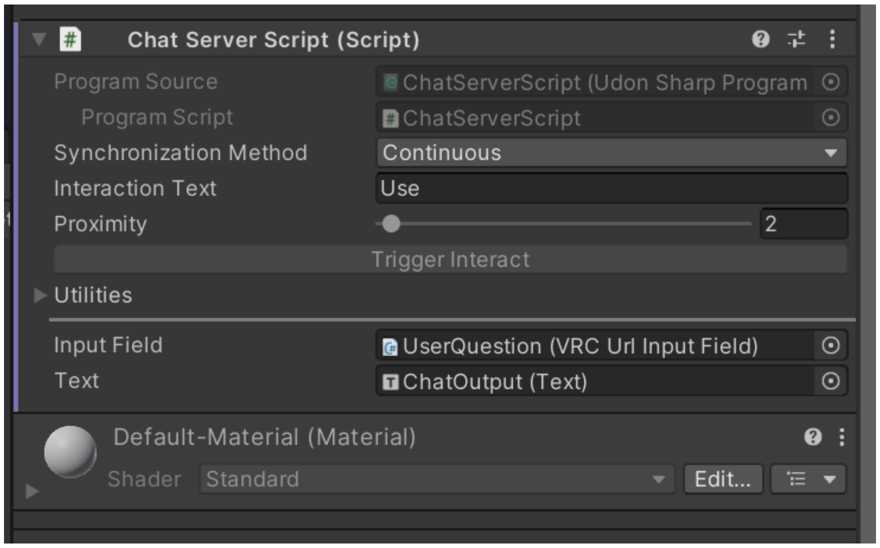
ii. For On End Edit (String): Click the “+” sign and drag the “UserQuestion” itself from hierarchy to the box below Runtime. Then, Choose the VRCURLInputField>SendMessage



Placeholder: Placeholder (Text)
Caret Blink Rate
Caret Width
Custom Caret Color
Selection Color
Hide Mobile Input
Read Only
On Value Changed (String)
Runtime Only — Text.text (string)
Placeholder (T
On End Edit (String)
Runtime Only — No Function
UserQuestion
VRC Ui Shape (Sc
Script
Default UI Materi
Shader UI/Defaul
Sprite Texture
Tiling X 1 Y 1
Offset X 0 Y 0
Tint
Stencil Comparison
Stencil ID
Stencil Operation
UserQuestion

iv. Making the required relations between the objects and UI:

1. Step 4: Adding in the Utilites in the Inspector of ChatButtonCube's Chat Server Script. .
   a. Drag the "ChatOutput" from Hierachy and release (place) on the "Text" in the Utilites in the Inspector.
   b. Drag the "UserQuestion" from the Hierarchy and release on the "InputField" in the Utilites in the Inspector.



Chat Server Script (Script)
Program Source: ChatServerScript (Udon Sharp Program
Program Script: ChatServerScript
Synchronization Method: Continuous
Interaction Text: Use
Proximity: 2
Utilities
Input Field: UserQuestion (VRC Url Input Field)
Text: ChatOutput (Text)
Default-Material (Material)
Shader: Standard

3. Server Connection (FastAPI)

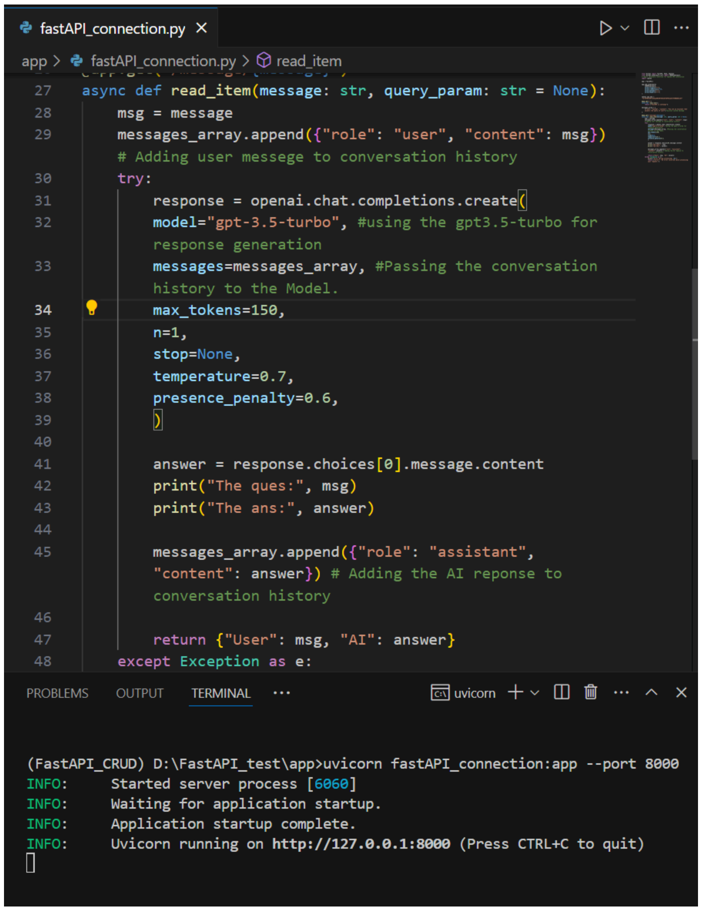
a. To create a FastAPI server:

i. Step 1: Create a python file and name it fastApi_connection.py and copy the following code inside that file:

```
from fastapi import FastAPI, Body, Depends
from fastapi.middleware.cors import CORSMiddleware
# Import necessary libraries for OpenAI API interaction
import openai

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

openai.api_key = "Place you Actual Open AI Api key here"

@app.get("/")
async def read_root():
    return {"fastAPI is working!"}

messages_array = [
    {"role": "system", "content": "You are an Assitant
that answers the query of user"} #initial system messege
]

@app.get("/message/{message}")
async def read_item(message: str, query_param: str =
None):
    msg = message
    messages_array.append({"role": "user", "content":
msg}) # Adding user messege to conversation history
    try:
        response = openai.chat.completions.create(
        model="gpt-3.5-turbo", #using the gpt3.5-turbo
for response generation
        messages=messages_array, #Passing the
conversation history to the Model.
```

```
        max_tokens=150,
        n=1,
        stop=None,
        temperature=0.7,
        presence_penalty=0.6,
        )

        answer = response.choices[0].message.content
        print("The ques:", msg)
        print("The ans:", answer)

        messages_array.append({"role": "assistant",
"content": answer}) # Adding the AI reponse to
conversation history

        return {"User": msg, "AI": answer}
    except Exception as e:
        print(f"Error during processing: {e}")
        return {"error": "An error occurred while
processing your request."}
```

ii. Step 2: Install required libraries such as openai, fastapi and uvicorn (ie: **pip install openai==1.16.2 fastapi uvicorn**)
iii. Step 3: On the terminal/CMD run this file by typing **uvicorn fastAPI_connection:app --port 8000.** This will setup a localserver that listens on localhost:8000 port.



```python
async def read_item(message: str, query_param: str = None):
    msg = message
    messages_array.append({"role": "user", "content": msg})
    # Adding user messege to conversation history
    try:
        response = openai.chat.completions.create(
        model="gpt-3.5-turbo", #using the gpt3.5-turbo for
        response generation
        messages=messages_array, #Passing the conversation
        history to the Model.
        max_tokens=150,
        n=1,
        stop=None,
        temperature=0.7,
        presence_penalty=0.6,
        )

        answer = response.choices[0].message.content
        print("The ques:", msg)
        print("The ans:", answer)

        messages_array.append({"role": "assistant",
        "content": answer}) # Adding the AI reponse to
        conversation history

        return {"User": msg, "AI": answer}
    except Exception as e:
```

```
(FastAPI_CRUD) D:\FastAPI_test\app>uvicorn fastAPI_connection:app --port 8000
INFO:     Started server process [6060]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

4. Ready To Use in Unity/VrChat:

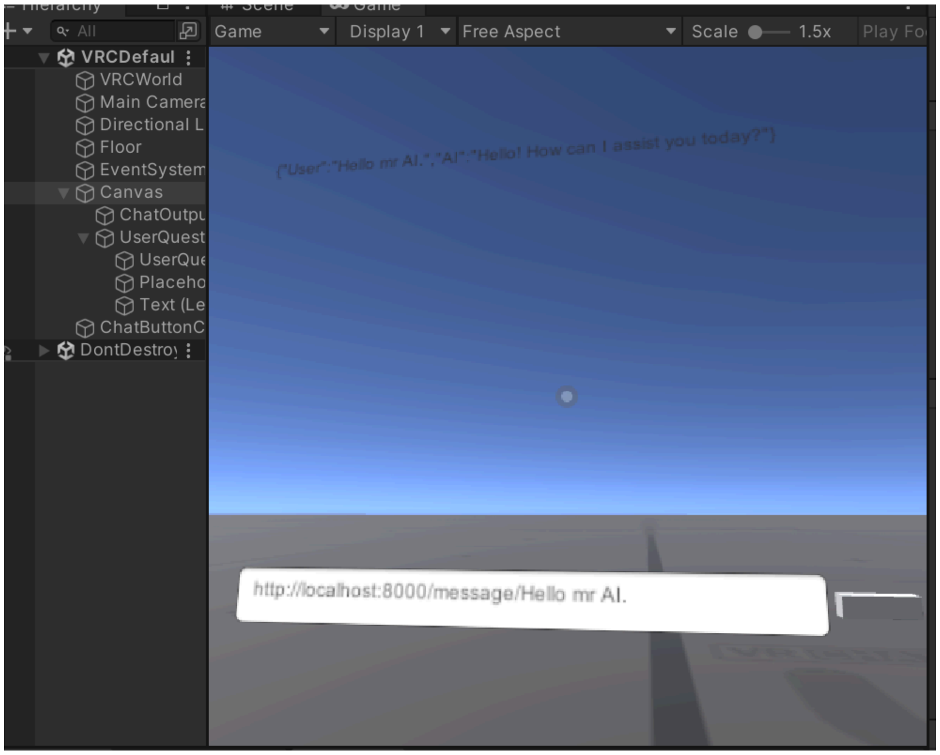
a. IN Unity , press the Play button on the “Game” window to start the demo.

i. Type in “ http://localhost:8000/message/Hello MR AI” in the text chat box. Then click the Cube Box.

ii. This will treat “Hello MR AI” as a message.

iii. On pressing the Cube Box, the message from text box will get passed to FastAPI server, where the OpenAI chat response is generated and passed back to the ChatOutput object in the Unity.

iv. The output will be as follows:



{"User":"Hello mr AI.","AI":"Hello! How can I assist you today?"}

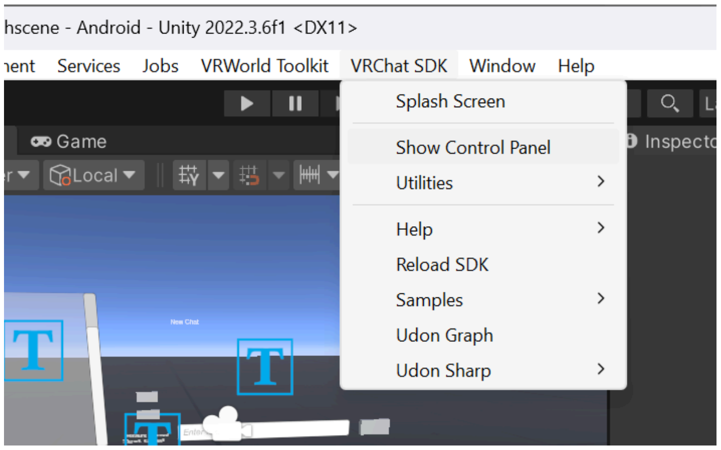
http://localhost:8000/message/Hello mr AI.

v. You can build and upload this to your VRChat account.

5. Chat Deploy in VRCHAT (with temporary public link)

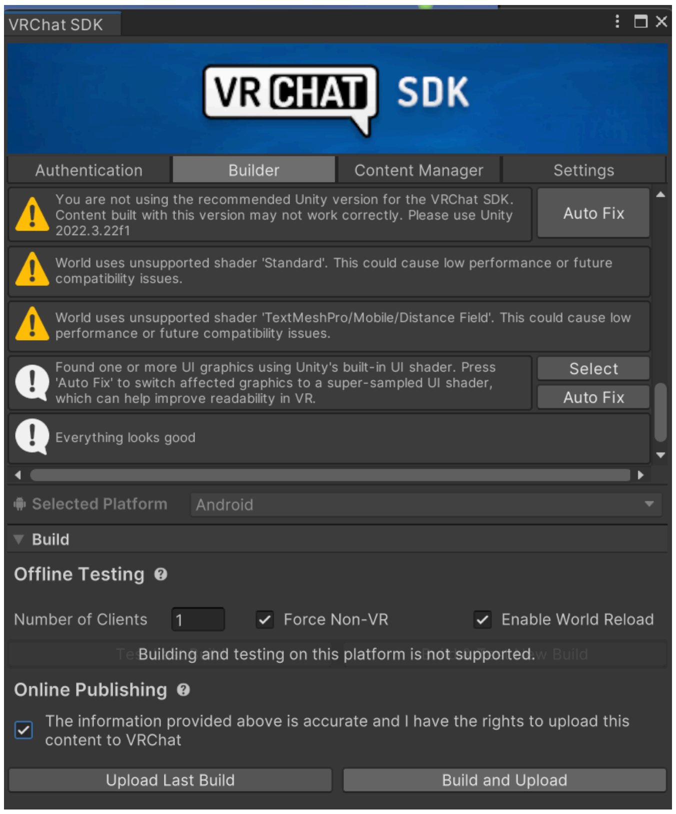
The created World now is ready for Build and Upload:
To upload in the VRChat follow the steps below:

1. Hover over the "VRChatSDK" in the Unity nav bar.
2. Click on the Show Control Panel



3. In the control panel, login if you are not already authorized.
4. Once login is complete, click on the Builder tab. And in the Builder tab:
   a. Choose appropriate thumbnail for the world you created.

b. Then put tick check mark on the Online Publishing
c. Then click on Build and Upload.
d. This will upload the VRChat you created to your VRChat account, which can be accessed from VRChat.



You are not using the recommended Unity version for the VRChat SDK. Content built with this version may not work correctly. Please use Unity 2022.3.22f1

World uses unsupported shader 'Standard'. This could cause low performance or future compatibility issues.

World uses unsupported shader 'TextMeshPro/Mobile/Distance Field'. This could cause low performance or future compatibility issues.

Found one or more UI graphics using Unity's built-in UI shader. Press 'Auto Fix' to switch affected graphics to a super-sampled UI shader, which can help improve readability in VR.

Everything looks good

Selected Platform: Android

Build

Offline Testing

Number of Clients: 1 ✓ Force Non-VR ✓ Enable World Reload

Building and testing on this platform is not supported.

Online Publishing

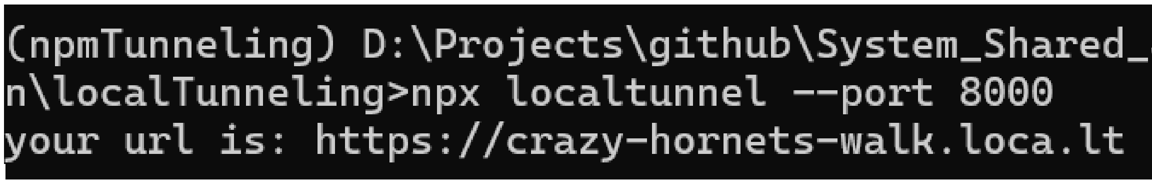
✓ The information provided above is accurate and I have the rights to upload this content to VRChat

However, Before Uploading the World please make sure you follow the below things so that the URL links that you put for messaging is accessible from anywhere not just the localhost. For this we can use “LocalTunneling” or “Ngrok”

a. **OPTION 1: USING “LOCALTUNNELING” (Recommended as give short url and more consistent)**
   i. Making the Chat function temporary available outside the localhost to use in VRCHAT.
   ii. Right now, the above solution only lets the chat to be utilized by the one within the localhost PC. Now to make it available for others we can use a free tunneling service provided by https://theboroer.github.io/localtunnel-www/
   iii. For this to work : We need to install node js (npm is included), first from here “https://nodejs.org/en”. Download and run the installer.
   iv. Once nodejs is install, then you need to install the **localtunnel** globally or locally as both works:
   v. Install LocalTunnell globally

1. Type this command in cmd terminal:
   npm install -g localtunnel
2. Then type this command in cmd
   lt --port 8000
3. Once, you type it, the tunnel will start and the URL to tunnel will be given and our FastAPI file running in localhost:8000 will be tunneled to the provided URL link. Users can use that link to send message.

vi. ALternatively, install LocalTunnel locally:
1. Navigate to your project directory in your terminal. Initialize a new Node.js project (if you haven't already) by running:
   npm init -y
2. Install Localtunnel as a local dependency by running:
   npm install localtunnel
3. After installing Localtunnel locally, you can use it from your project's node_modules directory. You can run Localtunnel commands using npx, which allows you to run Node.js binaries without installing them globally.
4. For example: npx localtunnel --port 8000



```
(npmTunneling) D:\Projects\github\System_Shared_
n\localTunneling>npx localtunnel --port 8000
your url is: https://crazy-hornets-walk.loca.lt
```

5. Once, you type it, the tunnel will start and the URL to pass request tunnel to our FastAPI file running in localhost:8000
6. Open your VRChat account and get to this world. Then inside you can chat with the AI, using the input field but everytime you need to write the
   “UrlGivenByLocalTunnel/message/your_actual_message.”

For e.g: https://tame-jokes-repair.loca.lt/message/Hello MR AI
…. The “Hello MR AI” is treated as the actual message.

7. Click the cube button on the side to send this message to AI, which in return will give response on the output field just as before.

b. OPTION 2: USING NGROK (Not recommended due to long url)

i. **NGROK.**
1. Step 1: Download the zip file of Ngrok from here https://ngrok.com/download.
2. Extract it and run it. This opens a CMD terminal where you should:
   a. Type in your authentication token that you get from ngrok as follows:
      ngrok config add-authtoken <token>
   b. Start tunnel by typing:

ngrok http 8000

ii. This will provide a temporary link tunnel to our fastAPI being hosted in 8000 port previously, that can be accessed from anywhere from public internet.
    1. We need to note down the public ip address given by ngrok as this should be typed in during chatting from the Unity or VRchat.

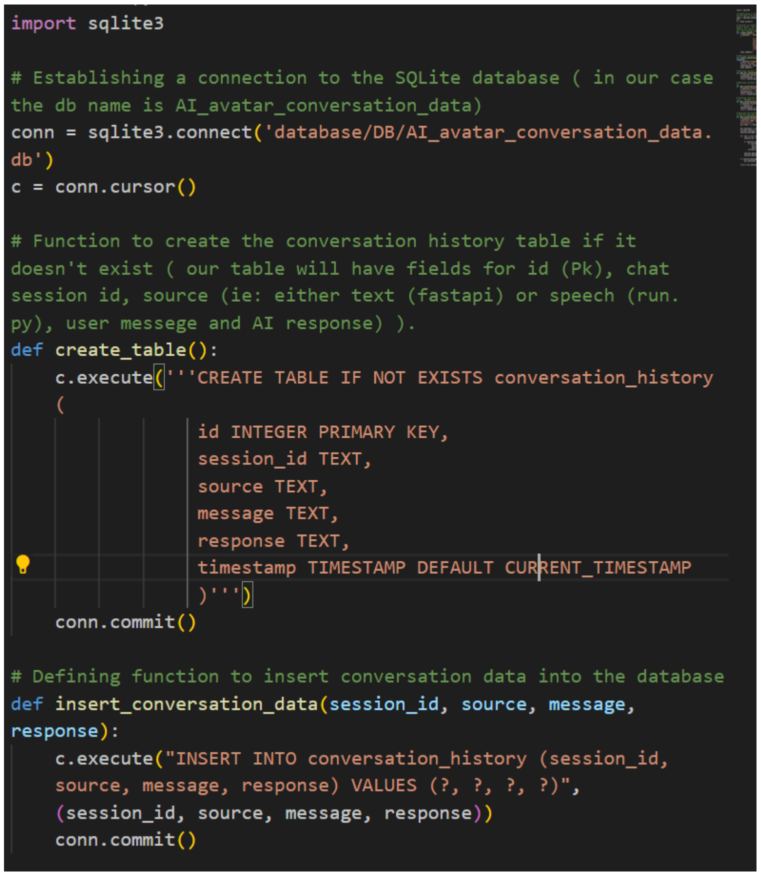
iii. Open your VRChat account and get to this world. Then inside you can chat with the AI, using the input field but everytime you need to write the "**ipaddress_given_by_ngrox/message/your_actual_message.**"
For e.g:
https://3d99-2400-1a00-b070-32bd-19bd-e4f4-ef75-96b6.ngrok-free.app/message/Hello MR AI
…. The "Hello MR AI" is treated as the actual message.

iv. Click the cube button on the side to send this message to AI, which in return will give response on the output field just as before.

6. Further Enhancements:

a. Setting up a database for message history. (Completed)

i. We can use Sqlte3 database to store the chat history for both speech and text chat based on Sessions using the code similar to this one :

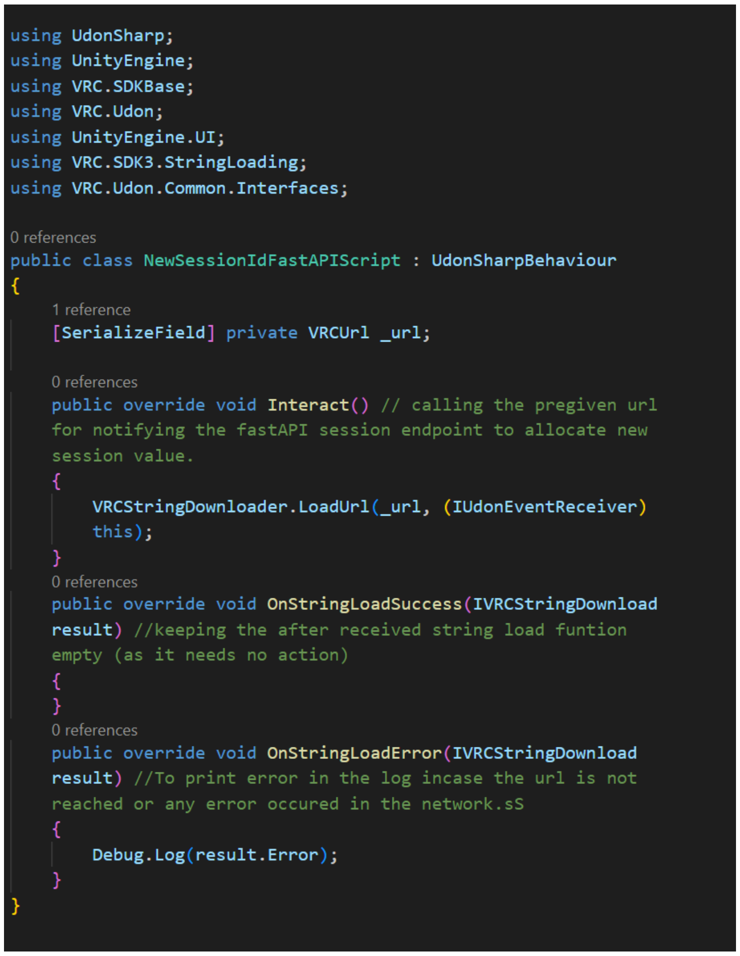


```
import sqlite3

# Establishing a connection to the SQLite database ( in our case
the db name is AI_avatar_conversation_data)
conn = sqlite3.connect('database/DB/AI_avatar_conversation_data.
db')
c = conn.cursor()

# Function to create the conversation history table if it
doesn't exist ( our table will have fields for id (Pk), chat
session id, source (ie: either text (fastapi) or speech (run.
py), user messege and AI response) ).
def create_table():
    c.execute('''CREATE TABLE IF NOT EXISTS conversation_history
    (
                id INTEGER PRIMARY KEY,
                session_id TEXT,
                source TEXT,
                message TEXT,
                response TEXT,
                timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                )''')
    conn.commit()

# Defining function to insert conversation data into the database
def insert_conversation_data(session_id, source, message,
response):
    c.execute("INSERT INTO conversation_history (session_id,
    source, message, response) VALUES (?, ?, ?, ?)",
    (session_id, source, message, response))
    conn.commit()
```

ii. Now, we can call the “insert_conversation_data” function when any new chat session occurs. This will store the session id, source (either text or speech), message (user message) and response.

iii. Moreover, the new sessions can be created for both the text as well as speech chat as follows:

1. For Text chat:
    a. Create a new 3d Cube in unity (similar to one created before)
    b. Attach a new U# script named that passes the http request to FastAPI using the “string-loading” method as shown below:
    c. When User clicks the cube to which the below code is attached , the request is passed to the URL (ie: fastAPI endpoint url in our case)



```
using UdonSharp;
using UnityEngine;
using VRC.SDKBase;
using VRC.Udon;
using UnityEngine.UI;
using VRC.SDK3.StringLoading;
using VRC.Udon.Common.Interfaces;

0 references
public class NewSessionIdFastAPIScript : UdonSharpBehaviour
{
    1 reference
    [SerializeField] private VRCUrl _url;

    0 references
    public override void Interact() // calling the pregiven url
    for notifying the fastAPI session endpoint to allocate new
    session value.
    {
        VRCStringDownloader.LoadUrl(_url, (IUdonEventReceiver)
        this);
    }
    0 references
    public override void OnStringLoadSuccess(IVRCStringDownload
    result) //keeping the after received string load funtion
    empty (as it needs no action)
    {
    }
    0 references
    public override void OnStringLoadError(IVRCStringDownload
    result) //To print error in the log incase the url is not
    reached or any error occured in the network.sS
    {
        Debug.Log(result.Error);
    }
}
```

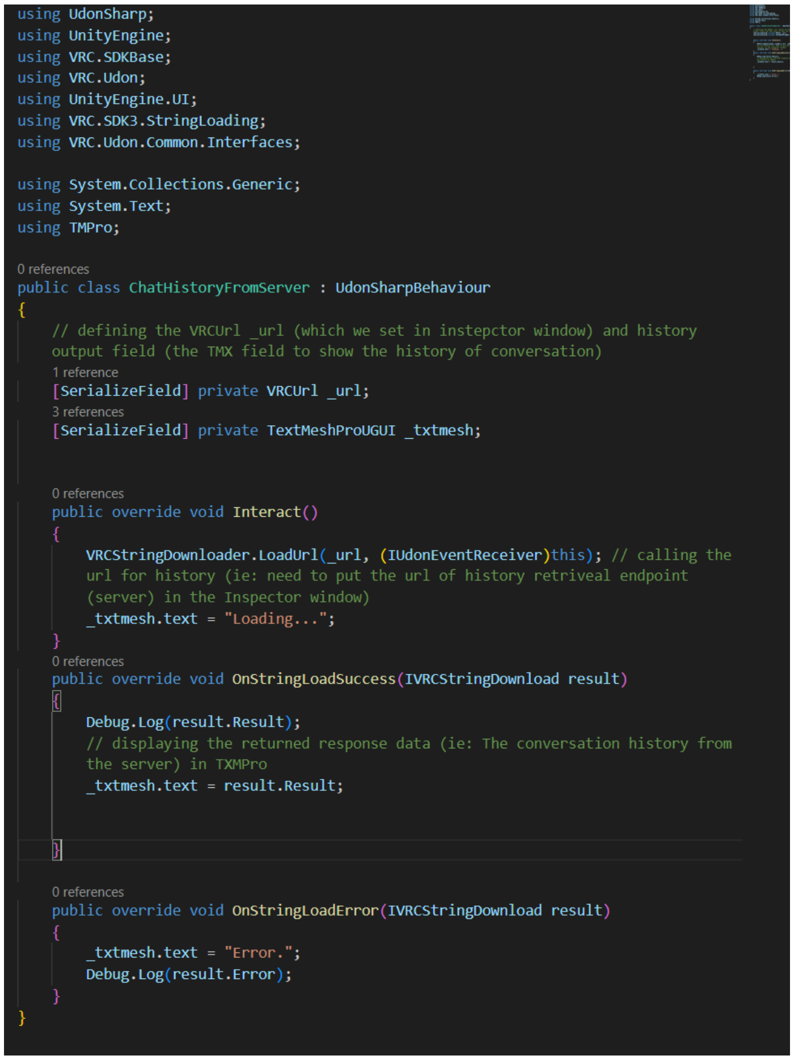
    d. Upon the request received by FastAPI, the fastAPI invokes a function that assings a new session id in the database for the next chat in the sequence until another new session.

2. For Speech chat:

a. New session for speech chat is created when the User in the VRChat says “Start New Session” to the AI Avatar.
b. For this, if the python code for speech recognition detects the “Start New Session” it will assign a new session id to the database.
c. Subsequent conversations are associated with this session ID until another "Start New Session" is heard.
d. The process is looped to continuously create new sessions upon detecting the phrase for a new session.

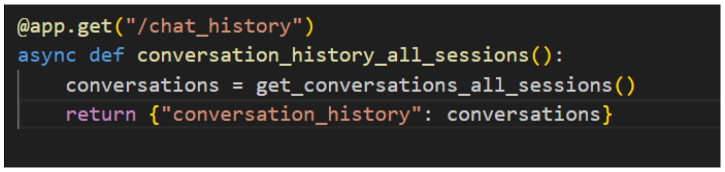
b. Reflect the message history in VRchat:

i. FastAPI endpoint can be called from VRChat using the String-Loader method using the UdonSharp Script:
For this: We need to create one U# file in unity and one FastAPI endpoint in our python backend.
in the unity create a new the UdonSharp file named `ChatHistoryFromServer”` and copy the below code

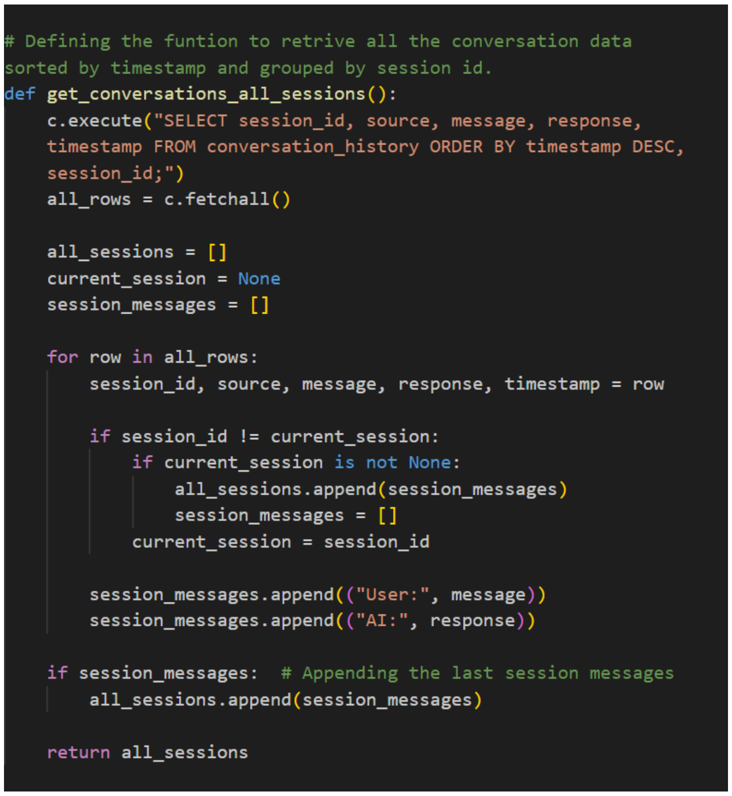


```
using UdonSharp;
using UnityEngine;
using VRC.SDKBase;
using VRC.Udon;
using UnityEngine.UI;
using VRC.SDK3.StringLoading;
using VRC.Udon.Common.Interfaces;

using System.Collections.Generic;
using System.Text;
using TMPro;

0 references
public class ChatHistoryFromServer : UdonSharpBehaviour
{
    // defining the VRCUrl _url (which we set in instepctor window) and history
    output field (the TMX field to show the history of conversation)
    1 reference
    [SerializeField] private VRCUrl _url;
    3 references
    [SerializeField] private TextMeshProUGUI _txtmesh;


    0 references
    public override void Interact()
    {
        VRCStringDownloader.LoadUrl(_url, (IUdonEventReceiver)this); // calling the
        url for history (ie: need to put the url of history retriveal endpoint
        (server) in the Inspector window)
        _txtmesh.text = "Loading...";
    }
    0 references
    public override void OnStringLoadSuccess(IVRCStringDownload result)
    {
        Debug.Log(result.Result);
        // displaying the returned response data (ie: The conversation history from
        the server) in TXMPro
        _txtmesh.text = result.Result;


    }

    0 references
    public override void OnStringLoadError(IVRCStringDownload result)
    {
        _txtmesh.text = "Error.";
        Debug.Log(result.Error);
    }
}
```

ii. Now we can attach this to a new 3d cube. In the inspector of this 3d cube put the URL of the FastAPI that calls the function for handling the extraction of data from database based on the sorted session and timestamp. And also, create and drag a new TMX Pro text field to display the history on it extracted from the server database.



```
@app.get("/chat_history")
async def conversation_history_all_sessions():
    conversations = get_conversations_all_sessions()
    return {"conversation_history": conversations}
```



```
# Defining the funtion to retrive all the conversation data
sorted by timestamp and grouped by session id.
def get_conversations_all_sessions():
    c.execute("SELECT session_id, source, message, response,
    timestamp FROM conversation_history ORDER BY timestamp DESC,
    session_id;")
    all_rows = c.fetchall()

    all_sessions = []
    current_session = None
    session_messages = []

    for row in all_rows:
        session_id, source, message, response, timestamp = row

        if session_id != current_session:
            if current_session is not None:
                all_sessions.append(session_messages)
                session_messages = []
            current_session = session_id

        session_messages.append(("User:", message))
        session_messages.append(("AI:", response))

    if session_messages:  # Appending the last session messages
        all_sessions.append(session_messages)

    return all_sessions
```

iii. Subsequently, the data will be extracted and passed back as a response to the U# Script that invoked this request to fast api using the VRCUrl and string-loading functions.

iv. The result variable of IVRCStringDownload type in the U# consist of the response text in json, that is then displayed using the TMXPro text field.

c. Sending the messages without typing the hostname along the message.(not completed)

d. Continue chatting from previous points. (not complete)