```
from flask import Flask, request, jsonify
from flask_cors import CORS
import requests
import re
import os

app = Flask(__name__)
CORS(app, resources={r"/*": {"origins": "*"}}, supports_credentials=True)

# ---- CONFIG ----
API_KEY = os.getenv(
    "OPENROUTER_KEY",
    "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",
)
URL = "https://openrouter.ai/api/v1/chat/completions"

# In-memory DB { email: {"username":..., "password":..., "history": [...] } }
users = {}

# Religion → language mapping (10 religions)
RELIGION_LANG = {
    "Hinduism": "hi",
    "Christianity": "en",
    "Islam": "ar",
```

```
    "Buddhism": "pi",
    "Sikhism": "pa",
    "Judaism": "he",
    "Bahá’í Faith": "fa",
    "Jainism": "gu",
    "Shinto": "ja",
    "Taoism / Daoism": "zh",
}

# ---- UTILITIES ----
def clean_response(text: str) -> str:
    """Remove unwanted tokens / symbols and take only the first paragraph."""
    if not text:
        return "कोई सुझाव नहीं मिला।"
    text = re.sub(r"◁think▷.*?◁/think▷", "", text, flags=re.DOTALL)
    text = re.sub(r"\$\s*\d+\s*:", "", text)
    text = re.sub(r"^\s*[\d]+\s*[:.]\s*", "", text, flags=re.MULTILINE)
    text = text.replace("$", "")
    text = re.sub(r"\n{2,}", "\n", text)
    # Keep only first paragraph (before first double newline)
    text = text.strip().split("\n")[0]
    return text.strip()

def get_openrouter_response(conflict_description, religion, language, format_request=None):
    """Ask OpenRouter for forgiveness advice in a single language and optional format."""
```

```
    prompt = (
        f"Always respond in a forgiving, gentle, and empathetic tone.\n"
        f"If the user sends greetings or off-topic messages, respond in a forgiving way and ask 'How can I help you today?'.\n"
        f"Conflict: {conflict_description}\n"
        f"Religion: {religion}\n"
        f"Respond only in {language} using the cultural tone of {religion}.\n"
        f"Do not include code, stories, or mixed languages.\n"
        f"Adjust the answer length based on the length of the conflict description.\n"
    )

    if format_request:
        prompt += f"Provide the answer in the format requested by the user: {format_request}\n"
    else:
        prompt += "Answer naturally in one paragraph.\n"

    payload = {
        "model": "z-ai/glm-4.5-air:free",
        "messages": [{"role": "user", "content": prompt}],
        "stream": False,
    }
    headers = {"Authorization": f"Bearer {API_KEY}", "Content-Type": "application/json"}
    try:
        r = requests.post(URL, json=payload, headers=headers, timeout=20)
        if r.status_code == 200:
            data = r.json()
```

```
            raw = data.get("choices", [{}])[0].get("message", {}).get("content", "")
            return clean_response(raw)
        return f"API error {r.status_code}: {r.text}"
    except Exception as e:
        return f"Request failed: {e}"

# ---- ROUTES ----
@app.route("/signup", methods=["POST"])
def signup():
    data = request.get_json()
    username = data.get("username", "").strip()
    email = data.get("email", "").strip()
    password = data.get("password", "").strip()

    if not username or not email or not password:
        return jsonify({"error": "All fields are required"}), 400
    if email in users:
        return jsonify({"error": "Email already exists"}), 400

    users[email] = {"username": username, "password": password, "history": []}
    return jsonify({"message": "Signup successful"}), 200

@app.route("/login", methods=["POST"])
def login():
    data = request.get_json()
    email = data.get("email", "").strip()
```

```
    password = data.get("password", "").strip()
    u = users.get(email)
    if not u or u["password"] != password:
        return jsonify({"error": "Invalid email or password"}), 400
    return jsonify({"username": u["username"], "email": email, "history": u["history"]})

@app.route("/get_advice", methods=["POST"])
@app.route("/get_advice", methods=["POST"])
def get_advice():
    data = request.get_json()
    email = data.get("email", "").strip()
    conflict = data.get("conflict_description", "").strip()
    religion = data.get("religion", "").strip()
    format_request = data.get("answer_format", None)

    if not email or email not in users:
        return jsonify({"error": "Invalid user"}), 400

    if not conflict:
        return jsonify({
            "advice": "Hello! Please describe your conflict, so I can give you forgiveness advice.",
            "history": users[email]["history"],
        })

    if religion not in RELIGION_LANG:
        return jsonify({"error": "Unsupported religion"}), 400
```

```
    # ✅ pick language sent from app, fallback to map
    client_lang = data.get("language") or RELIGION_LANG.get(religion, "en")

    advice = get_openrouter_response(conflict, religion, client_lang, format_request)

    chat_item = {
        "title": conflict[:25] + "...",
        "religion": religion,
        "messages": [
            {"message": conflict, "sender": "user"},
            {
                "message": advice,
                "sender": "AI",
                "original": advice,
                "translations": {},
                "currentLang": client_lang,
                "isTranslating": False,
            },
        ],
    }
    users[email]["history"].append(chat_item)
    return jsonify({"advice": advice, "history": users[email]["history"]})

@app.route("/translate", methods=["POST"])
def translate():
```

```
data = request.get_json()
text = data.get("text", "").strip()
target = data.get("target_lang", "en").strip()
if not text:
    return jsonify({"error": "No text to translate"}), 400

payload = {
    "model": "z-ai/glm-4.5-air:free",
    "messages": [
        {
            "role": "user",
            "content": f"Translate the following text into pure {target}, keep meaning same. "
                       f"Just give translation, no extra explanation:\n{text}",
        }
    ],
    "stream": False,
}
headers = {"Authorization": f"Bearer {API_KEY}", "Content-Type": "application/json"}
try:
    r = requests.post(URL, json=payload, headers=headers, timeout=20)
    if r.status_code == 200:
        data = r.json()
        translated = data.get("choices", [{}])[0].get("message", {}).get("content", "")
        return jsonify({"translated": clean_response(translated)})
    return jsonify({"error": f"Translation API error {r.status_code}"}), 500
except Exception as e:
```

```
        return jsonify({"error": f"Translation failed: {e}"}), 500


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)this is my main.py,import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'package:video_player/video_player.dart';
import 'package:flutter/services.dart';

const String baseUrl = "http://10.175.34.145:5000";

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'AI Forgiveness Recommender',
      theme: ThemeData.dark().copyWith(
        primaryColor: Colors.black,
        scaffoldBackgroundColor: Colors.black,
        textTheme: TextTheme(bodyMedium: TextStyle(color: Colors.white)),
      ),
      home: LoginScreen(),
    );
```

```
  }
}

/// ---------- Video Widget ----------
class LoopingVideo extends StatefulWidget {
  final String asset;
  final double width;
  final double height;

  const LoopingVideo({
    required this.asset,
    this.width = 80,
    this.height = 80,
    Key? key,
  }) : super(key: key);

  @override
  State<LoopingVideo> createState() => _LoopingVideoState();
}

class _LoopingVideoState extends State<LoopingVideo> {
  late VideoPlayerController _controller;

  @override
  void initState() {
    super.initState();
```

```
  _controller = VideoPlayerController.asset(widget.asset)
    ..initialize().then((_) {
      setState(() {});
      _controller.setLooping(true);
      _controller.play();
    });
}

@override
void dispose() {
  _controller.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return _controller.value.isInitialized
      ? SizedBox(
          width: widget.width,
          height: widget.height,
          child: VideoPlayer(_controller),
        )
      : SizedBox(
          width: widget.width,
          height: widget.height,
          child: CircularProgressIndicator(),
```

```
        );
  }
}

/// ---------------------- Login Screen ----------------------
class LoginScreen extends StatefulWidget {
  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final emailCtrl = TextEditingController();
  final passCtrl = TextEditingController();
  bool _loading = false;

  Future<void> loginUser() async {
    setState(() => _loading = true);
    try {
      final response = await http.post(
        Uri.parse("$baseUrl/login"),
        headers: {"Content-Type": "application/json"},
        body: jsonEncode({
          "email": emailCtrl.text.trim(),
          "password": passCtrl.text.trim(),
        }),
      );
```

```
  setState(() => _loading = false);

  if (response.statusCode == 200) {
    final data = Map<String, dynamic>.from(jsonDecode(response.body));
    final history = (data["history"] ?? [])
        .map<Map<String, dynamic>>((h) => Map<String, dynamic>.from(h))
        .toList();

    Navigator.pushReplacement(
      context,
      MaterialPageRoute(
        builder: (_) => AIChatScreen(
          username: data["username"] ?? "",
          email: data["email"] ?? "",
          history: history,
        ),
      ),
    );
  } else {
    final err = Map<String, dynamic>.from(jsonDecode(response.body));
    ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(err["error"] ?? "Login failed")));
  }
} catch (e) {
  setState(() => _loading = false);
  ScaffoldMessenger.of(context)
```

```
      .showSnackBar(SnackBar(content: Text("Network error: $e")));
  }
}

@override
Widget build(BuildContext context) => Scaffold(
    backgroundColor: Colors.black,
    body: Center(
      child: SingleChildScrollView(
        padding: EdgeInsets.all(20),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text("Login",
                style: TextStyle(fontSize: 24, color: Colors.white)),
            SizedBox(height: 20),
            TextField(
              controller: emailCtrl,
              decoration: InputDecoration(
                  hintText: "Email",
                  filled: true,
                  fillColor: Colors.grey[850]),
            ),
            SizedBox(height: 10),
            TextField(
              controller: passCtrl,
```

```
            obscureText: true,
            decoration: InputDecoration(
              hintText: "Password",
              filled: true,
              fillColor: Colors.grey[850]),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: _loading ? null : loginUser,
            child: _loading
              ? CircularProgressIndicator(color: Colors.white)
              : Text("Login"),
          ),
          TextButton(
            onPressed: () => Navigator.push(
              context,
              MaterialPageRoute(builder: (_) => SignupScreen()),
            ),
            child: Text("Create account"),
          ),
        ],
      ),
    ),
  ),
 );
}
```

```
/// ---------------------- Signup Screen ----------------------
class SignupScreen extends StatefulWidget {
  @override
  _SignupScreenState createState() => _SignupScreenState();
}

class _SignupScreenState extends State<SignupScreen> {
  final nameCtrl = TextEditingController();
  final emailCtrl = TextEditingController();
  final passCtrl = TextEditingController();
  bool _loading = false;

  Future<void> signupUser() async {
    setState(() => _loading = true);

    try {
      final response = await http.post(
        Uri.parse("$baseUrl/signup"),
        headers: {"Content-Type": "application/json"},
        body: jsonEncode({
          "username": nameCtrl.text.trim(),
          "email": emailCtrl.text.trim(),
          "password": passCtrl.text.trim(),
        }),
      );
```

```
    setState(() => _loading = false);

    if (response.statusCode == 200) {
      Navigator.pop(context);
      ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text("Signup successful! Please login.")));
    } else {
      final err = Map<String, dynamic>.from(jsonDecode(response.body));
      ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text(err["error"] ?? "Signup failed")));
    }
  } catch (e) {
    setState(() => _loading = false);
    ScaffoldMessenger.of(context)
        .showSnackBar(SnackBar(content: Text("Network error: $e")));
  }
}

@override
Widget build(BuildContext context) => Scaffold(
      backgroundColor: Colors.black,
      body: Center(
        child: SingleChildScrollView(
          padding: EdgeInsets.all(20),
          child: Column(
            mainAxisSize: MainAxisSize.min,
```

```
children: [
  Text("Signup",
     style: TextStyle(fontSize: 24, color: Colors.white)),
  SizedBox(height: 20),
  TextField(
    controller: nameCtrl,
    decoration: InputDecoration(
       hintText: "Name",
       filled: true,
       fillColor: Colors.grey[850]),
  ),
  SizedBox(height: 10),
  TextField(
    controller: emailCtrl,
    decoration: InputDecoration(
       hintText: "Email",
       filled: true,
       fillColor: Colors.grey[850]),
  ),
  SizedBox(height: 10),
  TextField(
    controller: passCtrl,
    obscureText: true,
    decoration: InputDecoration(
       hintText: "Password",
       filled: true,
```

```
                fillColor: Colors.grey[850]),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: _loading ? null : signupUser,
              child: _loading
                  ? CircularProgressIndicator(color: Colors.white)
                  : Text("Create Account"),
            ),
          ],
        ),
      ),
    ),
  );
}

/// ---------------------- AI Chat Screen ----------------------
/// ---------------------- AI Chat Screen ----------------------
class AIChatScreen extends StatefulWidget {
  final String username;
  final String email;
  final List<Map<String, dynamic>>? history;

  AIChatScreen({
    required this.username,
    required this.email,
```

```
    this.history,
  });

  @override
  _AIChatScreenState createState() => _AIChatScreenState();
}

class _AIChatScreenState extends State<AIChatScreen> {
  final List<String> religions = [
    "Hinduism",
    "Christianity",
    "Islam",
    "Sikhism",
    "Judaism",
    "Bahá’í Faith",
    "Jainism",
    "Shinto",
    "Taoism / Daoism"
  ];

  String selectedReligion = "Hinduism";
  final TextEditingController _controller = TextEditingController();
  final ScrollController _scrollController = ScrollController();
  final TextEditingController _searchCtrl = TextEditingController();
  bool _isLoading = false;
  bool _isSidebarMinimized = false;
```

```
List<Map<String, dynamic>> savedChats = [];
int currentChatIndex = -1;

Map<String, String> religionDefaultLang = {
  "Hinduism": "hi",
  "Christianity": "es",
  "Islam": "ar",
  "Sikhism": "pa",
  "Judaism": "he",
  "Bahá’í Faith": "fa",
  "Jainism": "gu",
  "Shinto": "ja",
  "Taoism / Daoism": "zh",
};

Map<String, List<String>> translationCycle = {
  "Hinduism": ["hi", "en"],
  "Christianity": ["es", "en"],
  "Islam": ["ar", "en"],
  "Sikhism": ["pa", "en"],
  "Judaism": ["he", "en"],
  "Bahá’í Faith": ["fa", "en"],
  "Jainism": ["gu", "en"],
  "Shinto": ["ja", "en"],
  "Taoism / Daoism": ["zh", "en"],
};
```

```
@override
void initState() {
  super.initState();
  if (widget.history != null && widget.history!.isNotEmpty) {
    savedChats =
        widget.history!.map((h) => Map<String, dynamic>.from(h)).toList();
    currentChatIndex = 0;
  }
}

List<Map<String, dynamic>> get currentChatHistory {
  if (currentChatIndex == -1 || savedChats.isEmpty) return [];
  final msgs = savedChats[currentChatIndex]["messages"] ?? [];
  return List<Map<String, dynamic>>.from(msgs);
}

void updateCurrentChatHistory(List<Map<String, dynamic>> newMessages) {
  if (currentChatIndex == -1) return;
  setState(() {
    savedChats[currentChatIndex]["messages"] = newMessages;
  });
}

void newChat() {
  setState(() {
```

```
    savedChats.insert(0, {
      "title": "New Chat",
      "messages": [],
      "religion": selectedReligion,
      "defaultLang": religionDefaultLang[selectedReligion] ?? "en",
    });
    currentChatIndex = 0;
  });
}

void _scrollToBottom() {
  Future.delayed(Duration(milliseconds: 200), () {
    if (_scrollController.hasClients) {
      _scrollController.jumpTo(_scrollController.position.maxScrollExtent);
    }
  });
}

Future<void> getAdvice(String conflict, String religion) async {
  if (conflict.trim().isEmpty || _isLoading) return;
  if (currentChatIndex == -1) newChat();

  // Add the user message
  final msgs = List<Map<String, dynamic>>.from(currentChatHistory);
  msgs.add({"message": conflict, "sender": "user"});
  msgs.add({"message": "", "sender": "AI", "thinking": true});
```

```
updateCurrentChatHistory(msgs);
_scrollToBottom();
setState(() => _isLoading = true);

try {
 // ✅ Pass religion language to backend so first advice is in that language
 final res = await http.post(
  Uri.parse("$baseUrl/get_advice"),
  headers: {"Content-Type": "application/json"},
  body: jsonEncode({
   "email": widget.email,
   "conflict_description": conflict,
   "religion": religion,
   "language": religionDefaultLang[religion] ?? "en",
  }),
 );

 // Remove the thinking bubble
 final newMsgs = List<Map<String, dynamic>>.from(
    savedChats[currentChatIndex]["messages"]);
 newMsgs.removeWhere((m) => m["thinking"] == true);
 updateCurrentChatHistory(newMsgs);

 if (res.statusCode == 200) {
  final data = jsonDecode(res.body);
  final advice = (data["advice"] ?? "").toString().trim();
```

```
    // 👇 Store original text, translations, and currentLang
    String display = "";
    for (int i = 0; i < advice.length; i++) {
      display += advice[i];
      final temp = List<Map<String, dynamic>>.from(currentChatHistory);
      if (temp.isEmpty || temp.last["sender"] != "AI") {
        temp.add({
          "message": display,
          "sender": "AI",
          "original": advice,
          "translations": {},
          "currentLang": religionDefaultLang[religion] ?? "en",
        });
      } else {
        temp.last["message"] = display;
      }
      updateCurrentChatHistory(temp);
      _scrollToBottom();
      await Future.delayed(const Duration(milliseconds: 25));
    }
  } else {
    newMsgs.add({"message": "Error: ${res.body}", "sender": "AI"});
    updateCurrentChatHistory(newMsgs);
  }
} catch (e) {
```

```
    final errMsgs = List<Map<String, dynamic>>.from(currentChatHistory)
      ..add({"message": "Network error: $e", "sender": "AI"});
    updateCurrentChatHistory(errMsgs);
  } finally {
    setState(() => _isLoading = false);
    _scrollToBottom();
  }
}

// Future<void> getAdvice(String conflictDescription, String religion) async {
//   if (conflictDescription.trim().isEmpty || _isLoading) return;

//   if (currentChatIndex == -1) newChat();

//   final updatedMessages = List<Map<String, dynamic>>.from(currentChatHistory);
//   updatedMessages.add({"message": conflictDescription, "sender": "user"});
//   updatedMessages.add({"message": "", "sender": "AI", "thinking": true});
//   updateCurrentChatHistory(updatedMessages);
//   _scrollToBottom();
//   setState(() => _isLoading = true);

//   try {
//     final response = await http.post(
//       Uri.parse("$baseUrl/get_advice"),
//       headers: {"Content-Type": "application/json"},
//       body: jsonEncode({
```

```
//       "email": widget.email,
//       "conflict_description": conflictDescription,
//       "religion": religion,
//       "language": religionDefaultLang[religion] ?? "en",
//     }),
//   );

//   final newMessages = List<Map<String, dynamic>>.from(
//       savedChats[currentChatIndex]["messages"] ?? []);
//   // remove thinking placeholder
//   newMessages.removeWhere((msg) => msg["thinking"] == true);
//   updateCurrentChatHistory(newMessages);

//   if (response.statusCode == 200) {
//     final data = Map<String, dynamic>.from(jsonDecode(response.body));
//     final advice = (data["advice"] ?? "").toString().trim();

//     // Add message gradually
//     String displayedText = "";
//     for (int i = 0; i < advice.length; i++) {
//       displayedText += advice[i];
//       if (currentChatIndex == -1) break;
//       final tempMessages =
//           List<Map<String, dynamic>>.from(currentChatHistory);
//       if (tempMessages.isEmpty || tempMessages.last["sender"] != "AI") {
//         tempMessages.add({"message": displayedText, "sender": "AI"});
```

```
//         } else {
//           tempMessages[tempMessages.length - 1]["message"] = displayedText;
//         }
//         updateCurrentChatHistory(tempMessages);
//         _scrollToBottom();
//         await Future.delayed(Duration(milliseconds: 30)); // typing speed
//       }
//     } else {
//       newMessages.add({"message": "Error: ${response.body}", "sender": "AI"});
//       updateCurrentChatHistory(newMessages);
//     }
//   } catch (e) {
//     final newMessages = List<Map<String, dynamic>>.from(
//         savedChats[currentChatIndex]["messages"] ?? []);
//     newMessages.add({"message": "Network error: $e", "sender": "AI"});
//     updateCurrentChatHistory(newMessages);
//   } finally {
//     setState(() => _isLoading = false);
//     _scrollToBottom();
//   }
// }

Future<void> toggleTranslation(int index) async {
  final chat = currentChatHistory[index];
  if (chat["sender"] != "AI") return;
```

```
final newMessages = List<Map<String, dynamic>>.from(currentChatHistory);
List<String> cycle = translationCycle[selectedReligion] ?? ["en"];
String currentLang = chat["currentLang"] ?? cycle[0];
int nextIndex = (cycle.indexOf(currentLang) + 1) % cycle.length;
String nextLang = cycle[nextIndex];

if (chat["translations"][nextLang] == null) {
  newMessages[index]["isTranslating"] = true;
  updateCurrentChatHistory(newMessages);

  try {
    final res = await http.post(
      Uri.parse("$baseUrl/translate"),
      headers: {"Content-Type": "application/json"},
      body: jsonEncode({
        "text": chat["original"],
        "email": widget.email,
        "target_lang": nextLang,
      }),
    );
    if (res.statusCode == 200) {
      final data = jsonDecode(res.body);
      newMessages[index]["translations"][nextLang] =
          data["translated"] ?? "";
    } else {
      newMessages[index]["translations"][nextLang] = "[Translation failed]";
```

```
    }
   } catch (e) {
    newMessages[index]["translations"][nextLang] = "[Network error]";
   }

   newMessages[index]["isTranslating"] = false;
  }

  newMessages[index]["message"] = newMessages[index]["translations"]
      [nextLang] ??
    newMessages[index]["original"];
  newMessages[index]["currentLang"] = nextLang;

  updateCurrentChatHistory(newMessages);
 }

 @override
 Widget build(BuildContext context) {
  final chatHistory = currentChatHistory;

  return Scaffold(
   body: Row(
    children: [
     /// ------------------ Sidebar ------------------
     AnimatedContainer(
      duration: Duration(milliseconds: 300),
```

```
width: _isSidebarMinimized ? 70 : 250,
color: Colors.grey[900],
child: Column(
  children: [
    /// --------- 1. Top Bar ---------
    Container(
      color: Colors.black87,
      padding: EdgeInsets.symmetric(horizontal: 8, vertical: 8),
      child: Row(
        children: [
          LoopingVideo(
              asset: "assets/video1.mp4", width: 40, height: 40),
          Spacer(),
          IconButton(
            icon: Icon(
              _isSidebarMinimized
                  ? Icons.arrow_right
                  : Icons.arrow_left,
              color: Colors.white,
            ),
            onPressed: () {
              setState(
                  () => _isSidebarMinimized = !_isSidebarMinimized);
            },
          ),
        ],
```

```
  ),
),

/// --------- 2. New Chat + Search ---------
Padding(
  padding: EdgeInsets.symmetric(horizontal: 8, vertical: 8),
  child: Column(
    children: [
      ListTile(
        leading:
            Icon(Icons.add_circle_outline, color: Colors.white),
        title: _isSidebarMinimized
            ? null
            : Text("New Chat",
                style: TextStyle(color: Colors.white)),
        onTap: newChat,
      ),
      if (!_isSidebarMinimized)
        TextField(
          controller: _searchCtrl,
          style: TextStyle(color: Colors.white),
          decoration: InputDecoration(
            prefixIcon:
                Icon(Icons.search, color: Colors.white70),
            hintText: "Search chat \nChats",
            hintStyle: TextStyle(color: Colors.white54),
```

```
          filled: true,
          fillColor: const Color.fromARGB(255, 49, 48, 48),
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(12),
            borderSide: BorderSide.none,
          ),
        ),
        onChanged: (val) {
          setState(() {}); // Rebuild to filter chat list
        },
      ),
    ],
  ),
),

/// --------- 3. Chats List ---------
Expanded(
  child: ListView.builder(
    itemCount: savedChats.length,
    itemBuilder: (context, i) {
      final chatTitle = savedChats[i]["title"] ?? "Untitled";
      if (_searchCtrl.text.isNotEmpty &&
          !chatTitle
              .toLowerCase()
              .contains(_searchCtrl.text.toLowerCase())) {
        return SizedBox.shrink();
```

```
      }
      return ListTile(
        leading: Icon(Icons.chat, color: Colors.white),
        title: _isSidebarMinimized
            ? null
            : Text(chatTitle,
                style: TextStyle(color: Colors.white)),
        onTap: () => setState(() => currentChatIndex = i),
      );
    },
  ),
),

/// --------- 4. User Info ---------
Container(
  color: Colors.grey[850],
  padding: EdgeInsets.symmetric(horizontal: 12, vertical: 12),
  child: Row(
    children: [
      CircleAvatar(
        backgroundColor: Colors.blue,
        child: Text(
          widget.username.isNotEmpty
              ? widget.username[0].toUpperCase()
              : "?",
          style: TextStyle(color: Colors.white),
```

```
            ),
          ),
          if (!_isSidebarMinimized) SizedBox(width: 12),
          if (!_isSidebarMinimized)
            Expanded(
              child: Text(widget.username,
                  style: TextStyle(color: Colors.white)),
            ),
        ],
      ),
    ),
  ],
 ),
),

/// ------------------ Main Chat ------------------
Expanded(
 child: Column(
  children: [
   /// Religion dropdown
   Container(
    alignment: Alignment.centerLeft,
    padding: EdgeInsets.all(12),
    child: DropdownButton<String>(
     value: selectedReligion,
     dropdownColor: Colors.grey[900],
```

```
    style: TextStyle(color: Colors.white),
    items: religions
        .map((r) => DropdownMenuItem(value: r, child: Text(r)))
        .toList(),
    onChanged: (val) {
      if (val != null) setState(() => selectedReligion = val);
    },
  ),
),

/// Chat History
Expanded(
  child: chatHistory.isEmpty
      ? Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              LoopingVideo(
                  asset: "assets/video3.mp4",
                  width: 120,
                  height: 120),
              SizedBox(height: 20),
              Text("Your road to peace begins here… 🕊️✨",
                  style: TextStyle(
                      color: Colors.white70,
                      fontSize: 18,
```

```
            fontWeight: FontWeight.w500)),
      SizedBox(height: 10),
      Text("Ask me anything about forgiveness",
        style: TextStyle(
          color: Colors.white54, fontSize: 14)),
    ],
   ),
  )
: ListView.builder(
   controller: _scrollController,
   itemCount: chatHistory.length,
   itemBuilder: (context, i) {
    final isUser = chatHistory[i]["sender"] == "user";
    final message =
      chatHistory[i]["message"]?.toString() ?? "";
    final currentLang =
      chatHistory[i]["currentLang"] ?? "";

    return Padding(
     padding: const EdgeInsets.symmetric(
       vertical: 6, horizontal: 120),
     child: Align(
      alignment: isUser
        ? Alignment.centerRight
        : Alignment.centerLeft,
      child: isUser
```

```
? Container(
    padding: EdgeInsets.all(12),
    decoration: BoxDecoration(
      color: Colors.blue[100],
      borderRadius:
          BorderRadius.circular(12),
    ),
    child: SelectableText(
      message,
      style: TextStyle(color: Colors.black),
    ),
  )
: chatHistory[i]["thinking"] == true
    ? Container(
        width: 60,
        height: 60,
        alignment: Alignment.center,
        child: CircularProgressIndicator(
          color: Colors.blueAccent,
          strokeWidth: 3,
        ),
      )
    : Column(
        crossAxisAlignment:
            CrossAxisAlignment.start,
        children: [
```

```
SelectableText(
  message,
  style: TextStyle(
      color: Colors.white,
      fontSize: 15),
),
SizedBox(height: 4),
Text(currentLang,
    style: TextStyle(
        color: Colors.white54,
        fontSize: 11)),
SizedBox(height: 6),
Row(
  children: [
    TextButton.icon(
      onPressed: () =>
          toggleTranslation(i),
      icon: Icon(Icons.translate,
          size: 18,
          color: Colors.white70),
      label: Text("Translate",
          style: TextStyle(
              color: Colors
                  .white70)),
    ),
    TextButton.icon(
```

```
                    onPressed: () {
                      Clipboard.setData(
                          ClipboardData(
                              text: message));
                      ScaffoldMessenger.of(
                              context)
                          .showSnackBar(SnackBar(
                              content: Text(
                                  "Copied to clipboard")));
                    },
                    icon: Icon(Icons.copy,
                        size: 18,
                        color: Colors.white70),
                    label: Text("Copy",
                        style: TextStyle(
                            color: Colors
                                .white70)),
                  ),
                ],
              ),
            ],
          ),
        ),
      );
    },
  ),
```

```
)

/// Input field
,
Padding(
  padding:
      const EdgeInsets.symmetric(horizontal: 120, vertical: 12),
  child: TextField(
    controller: _controller,
    style: TextStyle(color: Colors.white),
    decoration: InputDecoration(
      hintText: "Describe your conflict...",
      hintStyle: TextStyle(color: Colors.white54),
      filled: true,
      fillColor: Colors.grey[850],
      border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(12),
          borderSide: BorderSide.none),
      suffixIcon: IconButton(
        icon: Icon(Icons.send, color: Colors.blue),
        onPressed: _isLoading
            ? null
            : () {
                final text = _controller.text.trim();
                if (text.isNotEmpty) {
                  getAdvice(text, selectedReligion);
```

```
                        _controller.clear();
                      }
                    },
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    ],
  ),
 );
 }
}
```

this is my main.dart, now i want ai should ans only in english for all the 9 rleigions and add 1 more budhism in it and remove the tranaltion button from the chat section ,and give me updated copy patse code only i want full main.dart updated wokring copy paste code onlyyyyy only