package com.example.chatgptclient.logic.dao

import androidx.room.Dao

import androidx.room.Delete

import androidx.room.Insert

import androidx.room.Query

import com.example.chatgptclient.logic.model.Chat

@Dao

interface ChatDao {

@Insert

fun insertChat(chat: Chat): Long

@Query("select \* from chats")

fun loadAllChats(): List<Chat>

@Query("update chats set chatName = :newChatName where id = :id")

fun updateChatName(id: Long, newChatName: String): Int

@Query("delete from chats where id = :chatId")

fun deleteChatByChatId(chatId: Long): Int

@Query("delete from chats")

fun deleteAllChats()

}

package com.example.chatgptclient.logic.dao

import androidx.room.Dao

import androidx.room.Insert

import androidx.room.Query

import com.example.chatgptclient.logic.model.Msg

@Dao

interface MsgDao {

@Insert

fun insertMsg(msg: Msg): Long

@Query("select \* from messages where chatId = :chatId")

fun loadMsgs(chatId: Long): List<Msg>

@Query("delete from messages where chatId = :chatId")

fun deleteMessagesByChatId(chatId: Long): Int

@Query("delete from messages")

fun deleteAllMessages()

}

package com.example.chatgptclient.logic.dao

import com.example.chatgptclient.ChatGPTClientApplication

object SettingsDao {

fun getApiKey(): String {

return ChatGPTClientApplication.sharedPreferences.getString("api\_key","") ?: ""

}

fun getIsMultiTurnCon(): Boolean {

return ChatGPTClientApplication.sharedPreferences.getBoolean("enable\_continuous\_conversation",false)

}

fun getTemInt(): Int {

return ChatGPTClientApplication.sharedPreferences.getInt("temperature",10)

}

}

package com.example.chatgptclient.logic.model

import androidx.room.Entity

import androidx.room.PrimaryKey

@Entity(tableName = "chats")

data class Chat(var chatName: String) {

@PrimaryKey(autoGenerate = true)

var id: Long = 0

}

package com.example.chatgptclient.logic.model

import androidx.room.Entity

import androidx.room.PrimaryKey

@Entity(tableName = "messages")

data class Msg (var content: String, val type: Int, var chatId: Long? = null) {

@PrimaryKey(autoGenerate = true)

var id: Long = 0

companion object {

const val TYPE\_RECEIVED = 0

const val TYPE\_SENT = 1

}

}

package com.example.chatgptclient.logic

import android.content.Context

import androidx.room.Database

import androidx.room.Room

import androidx.room.RoomDatabase

import com.example.chatgptclient.logic.dao.ChatDao

import com.example.chatgptclient.logic.dao.MsgDao

import com.example.chatgptclient.logic.model.Chat

import com.example.chatgptclient.logic.model.Msg

@Database(version = 1, entities = [Chat::class, Msg::class], exportSchema = false)

abstract class AppDatabase : RoomDatabase() {

abstract fun chatDao(): ChatDao

abstract fun msgDao(): MsgDao

companion object {

private var instance: AppDatabase? = null

@Synchronized

fun getDatabase(context: Context): AppDatabase {

instance?.let {

return it

}

return Room.databaseBuilder(context.applicationContext,

AppDatabase::class.java,"app\_database")

.build().apply {

instance = this

}

}

}

}

package com.example.chatgptclient.logic

import androidx.lifecycle.liveData

import com.aallam.openai.api.BetaOpenAI

import com.aallam.openai.api.chat.\*

import com.aallam.openai.api.http.Timeout

import com.aallam.openai.api.model.ModelId

import com.aallam.openai.client.OpenAI

import com.aallam.openai.client.OpenAIConfig

import com.example.chatgptclient.ChatGPTClientApplication

import com.example.chatgptclient.logic.dao.SettingsDao

import com.example.chatgptclient.logic.model.Chat

import com.example.chatgptclient.logic.model.Msg

import kotlinx.coroutines.\*

import kotlinx.coroutines.flow.Flow

import kotlin.time.Duration.Companion.seconds

object Repository {

private var apiKey: String = ""

private var isMulTurnCon: Boolean = false

private var temInt: Int = 10

private var temDouble: Double? = null

private var config: OpenAIConfig? = null

private var openAI: OpenAI? = null

private val chatDao = AppDatabase.getDatabase(ChatGPTClientApplication.context).chatDao()

private val msgDao = AppDatabase.getDatabase(ChatGPTClientApplication.context).msgDao()

private val scope = CoroutineScope(Dispatchers.IO)

private var job: Job? = null

init {

job = scope.launch {

val deferredApiKey = async { SettingsDao.getApiKey() }

val deferredIsMulTurnCon = async { SettingsDao.getIsMultiTurnCon() }

val deferredTemInt = async { SettingsDao.getTemInt() }

apiKey = deferredApiKey.await()

isMulTurnCon = deferredIsMulTurnCon.await()

temInt = deferredTemInt.await()

temDouble = temInt.toDouble() / 10

config = OpenAIConfig(token = apiKey, timeout = Timeout(socket = 60.seconds))

openAI = OpenAI(config!!)

}

}

fun closeScope() {

job?.cancel()

}

fun setApiKey(apiKey: String) {

config = OpenAIConfig(token = apiKey, timeout = Timeout(socket = 60.seconds))

openAI = OpenAI(config!!)

}

fun setMulTurnCon(enable: Boolean) {

isMulTurnCon = enable

}

fun setTem(tem: Int) {

temDouble = tem.toDouble()/10

}

@OptIn(BetaOpenAI::class)

fun getChatCompletions(sendMsgs: List<ChatMessage>): Flow<ChatCompletionChunk> {

val model = ModelId("gpt-3.5-turbo")

val messages = if (isMulTurnCon) {

sendMsgs

} else {

listOf(sendMsgs.last())

}

val chatCompletionRequest = ChatCompletionRequest(

model,

messages,

temDouble

)

return openAI!!.chatCompletions(chatCompletionRequest)

}

suspend fun addNewChat(chat: Chat): Result<Long> {

return try {

withContext(Dispatchers.IO) {

val num = async { chatDao.insertChat(chat) }.await()

Result.success(num)

}

} catch (e: Exception) {

Result.failure(e)

}

}

suspend fun renameChatName(chatId: Long, chatName: String): Result<Int> {

return try {

withContext(Dispatchers.IO) {

val num = async { chatDao.updateChatName(chatId,chatName) }.await()

Result.success(num)

}

} catch (e: Exception) {

Result.failure(e)

}

}

suspend fun deleteChatAndMsgs(chatId: Long): Result<Int> {

return try {

withContext(Dispatchers.IO) {

val deferredChatNum = async { chatDao.deleteChatByChatId(chatId) }

val deferredMsgsNum = async { msgDao.deleteMessagesByChatId(chatId) }

deferredChatNum.await()

deferredMsgsNum.await()

Result.success(1)

}

} catch (e: Exception) {

Result.failure(e)

}

}

fun loadAllChats() = liveData(Dispatchers.IO) {

val result = try {

coroutineScope {

val chatList = async { chatDao.loadAllChats() }.await()

Result.success(chatList)

}

} catch (e: Exception) {

Result.failure(e)

}

emit(result)

}

suspend fun addMsg(msg: Msg): Result<Long> {

return try {

withContext(Dispatchers.IO) {

val num = msgDao.insertMsg(msg)

Result.success(num)

}

} catch (e: Exception) {

Result.failure(e)

}

}

suspend fun loadMsgsOfChat(chatId: Long): Result<List<Msg>> {

return try {

withContext(Dispatchers.IO) {

val msgList = async { msgDao.loadMsgs(chatId) }.await()

Result.success(msgList)

}

} catch (e: Exception) {

Result.failure(e)

}

}

suspend fun clearAllChatsAndMsgs(): Result<Int> {

return try {

withContext(Dispatchers.IO) {

val deferredChats = async { chatDao.deleteAllChats() }

val deferredMessages = async { msgDao.deleteAllMessages() }

deferredChats.await()

deferredMessages.await()

Result.success(1)

}

} catch (e: Exception) {

Result.failure(e)

}

}

}

package com.example.chatgptclient.ui.chat.chatlist

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import android.widget.TextView

import androidx.recyclerview.widget.RecyclerView

import com.example.chatgptclient.R

import com.example.chatgptclient.logic.model.Chat

import com.example.chatgptclient.ui.chat.ChatActivity

import com.example.chatgptclient.ui.chat.ChatViewModel

class ChatAdapter(private val chatActivity: ChatActivity, private val chatList: List<Chat>): RecyclerView.Adapter<ChatAdapter.ViewHolder>() {

inner class ViewHolder(view: View): RecyclerView.ViewHolder(view) {

val chatName: TextView = view.findViewById(R.id.chatName)

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {

val view = LayoutInflater.from(parent.context).inflate(R.layout.chat\_item,parent,false)

val holder = ViewHolder(view)

holder.itemView.setOnClickListener {

if (chatActivity.msgListViewModel.isSend.value == true) {

val position = holder.bindingAdapterPosition

val chat = chatList[position]

chatActivity.chatViewModel.chatName = chat.chatName

chatActivity.topAppBar.title = chatActivity.chatViewModel.chatName

ChatViewModel.curChatId = chat.id

chatActivity.chatViewModel.isChatGPT = false

chatActivity.msgRecyclerView.visibility = View.VISIBLE

chatActivity.bgTextView.visibility = View.GONE

chatActivity.chatViewModel.loadMsgsOfChat(chat.id)

}

}

return holder

}

override fun getItemCount(): Int {

return chatList.size

}

override fun onBindViewHolder(holder: ViewHolder, position: Int) {

val chat = chatList[position]

holder.chatName.text = chat.chatName

}

}

package com.example.chatgptclient.ui.chat.chatlist

import androidx.lifecycle.ViewModel

import com.example.chatgptclient.logic.model.Chat

class ChatListViewModel: ViewModel() {

val chatList = ArrayList<Chat>()

}

package com.example.chatgptclient.ui.chat.chatmain

import android.annotation.SuppressLint

import android.content.ClipData

import android.content.ClipboardManager

import android.content.Context

import android.graphics.Canvas

import android.graphics.Color

import android.graphics.Paint

import android.graphics.drawable.Drawable

import android.text.Layout

import android.text.Spanned

import android.text.TextPaint

import android.text.style.ClickableSpan

import android.text.style.LeadingMarginSpan

import android.text.util.Linkify

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import android.widget.TextView

import android.widget.Toast

import androidx.appcompat.content.res.AppCompatResources.getDrawable

import androidx.core.content.ContextCompat

import androidx.recyclerview.widget.LinearLayoutManager

import androidx.recyclerview.widget.RecyclerView

import androidx.recyclerview.widget.RecyclerView.ViewHolder

import com.example.chatgptclient.ChatGPTClientApplication

import com.example.chatgptclient.R

import com.example.chatgptclient.logic.model.Msg

import com.linhaodev.prism4jx.Prism4jGrammarLocator

import es.dmoral.toasty.Toasty

import io.noties.markwon.AbstractMarkwonPlugin

import io.noties.markwon.Markwon

import io.noties.markwon.MarkwonConfiguration

import io.noties.markwon.MarkwonSpansFactory

import io.noties.markwon.core.MarkwonTheme

import io.noties.markwon.ext.latex.JLatexMathPlugin

import io.noties.markwon.inlineparser.MarkwonInlineParserPlugin

import io.noties.markwon.linkify.LinkifyPlugin

import io.noties.markwon.recycler.MarkwonAdapter

import io.noties.markwon.recycler.SimpleEntry

import io.noties.markwon.syntax.Prism4jSyntaxHighlight

import io.noties.markwon.syntax.Prism4jThemeDefault

import io.noties.markwon.utils.LeadingMarginUtils

import io.noties.prism4j.Prism4j

import org.commonmark.node.FencedCodeBlock

class MsgAdapter(private val msgList: List<Msg>, textView: TextView): RecyclerView.Adapter<RecyclerView.ViewHolder>() {

private val markwon = Markwon.builder(ChatGPTClientApplication.context)

.usePlugin(MyPlugin())

.usePlugin(LinkifyPlugin.create(Linkify.WEB\_URLS))

.usePlugin(MarkwonInlineParserPlugin.create())

.usePlugin(JLatexMathPlugin.create(textView.textSize,MyJLatexMathPlugin()))

.build()

inner class LeftViewHolder(view: View): RecyclerView.ViewHolder(view) {

val leftMsgRecyclerView: RecyclerView = view.findViewById(R.id.leftMsg)

}

inner class RightViewHolder(view: View): RecyclerView.ViewHolder(view) {

val rightMsg: TextView = view.findViewById(R.id.rightMsg)

}

override fun getItemViewType(position: Int): Int {

val msg = msgList[position]

return msg.type

}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {

val holder: ViewHolder

if (viewType == Msg.TYPE\_RECEIVED) {

val view = LayoutInflater.from(parent.context).inflate(R.layout.msg\_left\_item,parent,false)

holder = LeftViewHolder(view)

} else {

val view = LayoutInflater.from(parent.context).inflate(R.layout.msg\_right\_item,parent,false)

holder = RightViewHolder(view)

}

return holder

}

override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {

val msg = msgList[position]

when (holder) {

is LeftViewHolder -> {

val markwonAapter: MarkwonAdapter = MarkwonAdapter.builderTextViewIsRoot(R.layout.adapter\_default\_entry)

.include(FencedCodeBlock::class.java, SimpleEntry.create(R.layout.adapter\_fenced\_code\_block, R.id.text))

.build()

val layoutManager = LinearLayoutManager(ChatGPTClientApplication.context)

holder.leftMsgRecyclerView.layoutManager = layoutManager

holder.leftMsgRecyclerView.adapter = markwonAapter

markwonAapter.setMarkdown(markwon, Regex("(?<!\\$)\\$(?!\\$)").replace(msg.content){ matchResult -> "$$" })

}

is RightViewHolder -> holder.rightMsg.text = msg.content

else -> throw IllegalAccessException()

}

}

override fun getItemCount(): Int {

return msgList.size

}

inner class MyPlugin : AbstractMarkwonPlugin() {

override fun configureTheme(builder: MarkwonTheme.Builder) {

builder.codeBlockTextColor(Color.WHITE)

builder.codeBlockBackgroundColor(Color.BLACK)

}

override fun configureConfiguration(builder: MarkwonConfiguration.Builder) {

val prism4j = Prism4j(Prism4jGrammarLocator())

val highlight = Prism4jSyntaxHighlight.create(prism4j, Prism4jThemeDefault.create())

builder.syntaxHighlight(highlight)

}

@SuppressLint("UseCompatLoadingForDrawables")

override fun configureSpansFactory(builder: MarkwonSpansFactory.Builder) {

builder.appendFactory(FencedCodeBlock::class.java) { \_, \_ ->

CopyContentsSpan()

}

builder.appendFactory(FencedCodeBlock::class.java) { \_, \_ ->

CopyIconSpan(getDrawable(ChatGPTClientApplication.context,R.drawable.ic\_code\_white\_24dp)!!)

}

}

}

class CopyContentsSpan() : ClickableSpan() {

override fun onClick(widget: View) {

val spanned = (widget as? TextView)?.text as? Spanned ?: return

val start = spanned.getSpanStart(this)

val end = spanned.getSpanEnd(this)

// by default code blocks have new lines before and after content

val contents = spanned.subSequence(start, end).toString().trim()

val clipboard = ChatGPTClientApplication.context.getSystemService(Context.CLIPBOARD\_SERVICE) as ClipboardManager

clipboard.setPrimaryClip(ClipData.newPlainText(null,contents))

Toasty.success(ChatGPTClientApplication.context, "复制成功", Toast.LENGTH\_SHORT, true).show()

}

override fun updateDrawState(ds: TextPaint) {

// do not apply link styling

}

}

class CopyIconSpan(val icon: Drawable) : LeadingMarginSpan {

init {

if (icon.bounds.isEmpty) {

icon.setBounds(0, 0, icon.intrinsicWidth, icon.intrinsicHeight)

}

}

override fun getLeadingMargin(first: Boolean): Int = 0

override fun drawLeadingMargin(

c: Canvas,

p: Paint,

x: Int,

dir: Int,

top: Int,

baseline: Int,

bottom: Int,

text: CharSequence,

start: Int,

end: Int,

first: Boolean,

layout: Layout

) {

// called for each line of text, we are interested only in first one

if (!LeadingMarginUtils.selfStart(start, text, this)) return

val save = c.save()

try {

// horizontal position for icon

val w = icon.bounds.width().toFloat()

// minus quarter width as padding

val left = layout.width - w - (w / 4F)

c.translate(left, top.toFloat())

icon.draw(c)

} finally {

c.restoreToCount(save)

}

}

}

inner class MyJLatexMathPlugin: JLatexMathPlugin.BuilderConfigure {

override fun configureBuilder(builder: JLatexMathPlugin.Builder) {

builder.let {

it.inlinesEnabled(true)

builder.theme().textColor(ContextCompat.getColor(ChatGPTClientApplication.context, R.color.royal\_blue));

}

}

}

}

package com.example.chatgptclient.ui.chat.chatmain

import androidx.lifecycle.MutableLiveData

import androidx.lifecycle.ViewModel

import androidx.lifecycle.viewModelScope

import com.aallam.openai.api.BetaOpenAI

import com.aallam.openai.api.chat.ChatMessage

import com.aallam.openai.api.chat.ChatRole

import com.example.chatgptclient.logic.Repository

import com.example.chatgptclient.logic.model.Msg

import com.example.chatgptclient.ui.chat.ChatViewModel

import kotlinx.coroutines.Dispatchers

import kotlinx.coroutines.flow.\*

import kotlinx.coroutines.launch

import kotlinx.coroutines.withContext

import java.util.\*

import kotlin.collections.ArrayList

@OptIn(BetaOpenAI::class)

class MsgListViewModel: ViewModel() {

private val \_msgContentResult = MutableStateFlow<Result<String>?>(null)

private val msgContentSB = StringBuilder()

private var count = 0

var isSend = MutableLiveData<Boolean>()

val msgList = ArrayList<Msg>()

val msgContentResult: StateFlow<Result<String>?> = \_msgContentResult.asStateFlow()

init {

isSend.value = true

}

fun sendMessage(message: String) {

viewModelScope.launch {

withContext(Dispatchers.IO) {

val sendMsgsList = getSendMsgsList(message)

Repository.getChatCompletions(sendMsgsList)

.catch { e ->

\_msgContentResult.value = Result.failure(e)

isSend.postValue(true)

}

.collect { chatCompletionChunk ->

chatCompletionChunk.choices[0].delta?.let {

if (it.role != null) {

val msg = Msg("", Msg.TYPE\_RECEIVED, ChatViewModel.curChatId)

msgList.add(msg)

msgContentSB.clear()

count = 0

} else {

count++

val content = it.content ?: ""

msgContentSB.append(content)

if (count == 5 || count == 25 || count == 50 || count % 80 == 0 || content == "") {

\_msgContentResult.value = Result.success(msgContentSB.toString())

}

}

}

if (chatCompletionChunk.choices[0].finishReason == "stop") {

isSend.postValue(true)

Repository.addMsg(Msg(msgContentSB.toString(), Msg.TYPE\_RECEIVED, ChatViewModel.curChatId))

}

}

}

}

}

/\*\*

\* 获取当前需要发送消息及历史消息

\*/

private fun getSendMsgsList(message: String): List<ChatMessage> {

val sendChatMessage = ChatMessage(

role = ChatRole.User,

content = message)

val sendMsgsList = mutableListOf(sendChatMessage)

var i = msgList.size - 2

var j = 2

loop@ while (i > 0 && j > 0) {

while (msgList[i].type == Msg.TYPE\_SENT) {

i--

if (i < 0) break@loop

}

if (msgList[i].type == Msg.TYPE\_RECEIVED) {

sendMsgsList.add(ChatMessage(

role = ChatRole.Assistant,

content = msgList[i].content

))

sendMsgsList.add(

ChatMessage(

role = ChatRole.User,

content = msgList[i-1].content

))

i -= 2

j--

}

}

sendMsgsList.reverse()

return sendMsgsList

}

}