```
const { SlashCommandBuilder, EmbedBuilder, ActionRowBuilder, ButtonBuilder,
ButtonStyle, ComponentType, MessageFlags } = require('discord.js');
const { getBalance, updateBalance } = require('../../utils/db');

// Helper: build and shuffle a deck of cards
const SUITS = ['♠', '♥', '♦', '♣'];
const RANKS = ['A','2','3','4','5','6','7','8','9','10','J','Q','K'];
function createShuffledDeck() {
  const deck = [];
  for (const suit of SUITS) {
    for (const rank of RANKS) {
      deck.push({ rank, suit });
    }
  }
  for (let i = deck.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [deck[i], deck[j]] = [deck[j], deck[i]];
  }
  return deck;
}
function drawCard(deck) {
  return deck.shift();
}
// Calculate best total for hand, counting Aces as 1 or 11
function calculateTotal(cards) {
  let total = 0;
  let aces = 0;
  for (const { rank } of cards) {
    if (rank === 'A') { aces++; total += 11; }
    else if (['J','Q','K'].includes(rank)) total += 10;
    else total += Number(rank);
  }
  while (total > 21 && aces > 0) {
    total -= 10;
    aces--;
  }
  return total;
}
function formatCards(cards) {
  return cards.map(c => `${c.rank}${c.suit}`).join(' ');
}
// Build embed showing hands, totals, bet, balance, and optional result
function buildEmbed(playerCards, dealerCards, balance, bet, result, payout) {
  const playerTotal = calculateTotal(playerCards);

  // Decide whether to show both dealer cards, or only the first one
  const inProgress = !result;
  const dealerDisplay = inProgress
    ? `${dealerCards[0].rank}${dealerCards[0].suit} ??`
    : formatCards(dealerCards);
```

```
  const dealerTotal = inProgress
    ? calculateTotal([dealerCards[0]])
    : calculateTotal(dealerCards);

  const embed = new EmbedBuilder()
    .setTitle('🎲 Blackjack')
    .addFields(
      { name: `Your Hand (Total: ${playerTotal})`, value: formatCards(playerCards),
inline: false },
      { name: `Dealer Hand (Total: ${dealerTotal})`, value: dealerDisplay, inline:
false },
      {
        name: 'Bet',
        value: payout !== undefined
          ? `${bet} (Won: ${payout})`
          : `${bet}`,
        inline: true
      },
      { name: 'Balance', value: String(balance), inline: true }
    );

  if (result) {
    if (result.startsWith('🃏')) {
      embed.setColor('#800080');        // Purple for Blackjack
    } else if (result.includes('2.5×')) {
      embed.setColor('#800080');        // Purple for 2.5× payout (blackjack)
    } else if (result.toLowerCase().includes('win')) {
      embed.setColor('#00FF00');        // Green for a standard win
    } else if (result.includes('Bust')) {
      embed.setColor('#FFCC00');        // Softer yellow for a Bust
    } else if (result.toLowerCase().includes('lose')) {
      embed.setColor('#FF0000');        // Red for a loss
    }
    embed.setDescription(`**${result}**`);
  }

  return embed;
}

module.exports = {
  data: new SlashCommandBuilder()
    .setName('bj')
    .setDescription('Play a round of Blackjack')
    .addSubcommand(sub => sub
      .setName('start')
      .setDescription('Start a new game')
      .addIntegerOption(opt => opt
        .setName('bet')
        .setDescription('Amount to bet')
        .setRequired(true)
```

```
      )
    ),

  async execute(interaction) {
    if (interaction.options.getSubcommand() !== 'start') return;
    const userId = interaction.user.id;
    let bet = interaction.options.getInteger('bet');
    const originalBet = bet;

    // Fetch and deduct initial bet
    const origBalance = await getBalance(userId);
    if (origBalance < bet) {
      return interaction.reply({ content: `💰 You only have ${origBalance}, you
cannot bet ${bet}.`, flags: MessageFlags.Ephemeral});
    }
    let balance = origBalance - bet;
    await updateBalance(userId, balance);

    // Function to run a full round, recursively called on "Play Again"
    const runRound = async () => {
      const deck = createShuffledDeck();
      let playerCards = [drawCard(deck), drawCard(deck)];
      let dealerCards = [drawCard(deck), drawCard(deck)];
      let firstMove = true;

      // Immediate Blackjack check
      if (calculateTotal(playerCards) === 21) {
        const payout = Math.floor(bet * 2.5);
        balance += payout;
        await updateBalance(userId, balance);
        const embed = buildEmbed(
          playerCards, dealerCards,
          balance, bet,
          '🃏 Blackjack! You win 2.5× your bet!',
          payout
        );
        const playRow = new ActionRowBuilder().addComponents(
          new ButtonBuilder()
            .setCustomId('play_again')
            .setLabel('Play Again')
            .setStyle(ButtonStyle.Primary)
        );
        const msg = await interaction.editReply({ embeds: [embed], components:
[playRow] });
        return handlePlayAgain(msg);
      }

      // Initial game embed with buttons
      const embed = buildEmbed(playerCards, dealerCards, balance, bet);
      const row = new ActionRowBuilder().addComponents(
```

```
        new
ButtonBuilder().setCustomId('hit').setLabel('Hit').setStyle(ButtonStyle.Success),
        new
ButtonBuilder().setCustomId('stand').setLabel('Stand').setStyle(ButtonStyle.Danger)
      );
      if (balance >= bet) {
        row.addComponents(
          new ButtonBuilder()
            .setCustomId('double')
            .setLabel(`Double Down (${bet * 2})`)
            .setStyle(ButtonStyle.Primary)
        );
      }

      // Send or update reply
      if (interaction.replied || interaction.deferred) {
        await interaction.editReply({ embeds: [embed], components: [row] });
      } else {
        await interaction.reply({ embeds: [embed], components: [row] });
      }

      // Fetch sent message and create collector
      const message = await interaction.fetchReply();
      const collector = message.createMessageComponentCollector({ componentType:
ComponentType.Button });

      collector.on('collect', async btnInt => {
        if (btnInt.user.id !== userId) {
          return btnInt.reply({ content: 'This isn’t your game!', flags:
MessageFlags.Ephemeral});
        }

        await btnInt.deferUpdate();
        const action = btnInt.customId;

        // HIT
        if (action === 'hit') {
          firstMove = false;
          playerCards.push(drawCard(deck));
          const total = calculateTotal(playerCards);
          if (total > 21) {
            // Bust
            collector.stop();
            const endEmbed = buildEmbed(playerCards, dealerCards, balance, bet,
'💥 Bust! You lose.');
            const playRow = new ActionRowBuilder().addComponents(
              new ButtonBuilder().setCustomId('play_again').setLabel('Play
Again').setStyle(ButtonStyle.Primary)
            );
            await message.edit({ embeds: [endEmbed], components: [playRow] });
```

```
            return handlePlayAgain(message);
          }
          // Update embed after hit
          const newEmbed = buildEmbed(playerCards, dealerCards, balance, bet);
          const newRow = new ActionRowBuilder().addComponents(
            new
ButtonBuilder().setCustomId('hit').setLabel('Hit').setStyle(ButtonStyle.Success),
            new
ButtonBuilder().setCustomId('stand').setLabel('Stand').setStyle(ButtonStyle.Danger)
          );
          if (firstMove && balance >= bet) {
            newRow.addComponents(
              new ButtonBuilder()
                .setCustomId('double')
                .setLabel(`Double Down (${bet * 2})`)
                .setStyle(ButtonStyle.Primary)
            );
          }
          return message.edit({ embeds: [newEmbed], components: [newRow] });
        }

        // DOUBLE DOWN
        if (action === 'double' && firstMove) {
          if (balance < bet) {
            return btnInt.followUp({ content: 'Insufficient balance to double
down.', flags: MessageFlags.Ephemeral});
          }
          // Deduct second bet
          balance -= bet;
          bet *= 2;
          await updateBalance(userId, balance);
          firstMove = false;
          playerCards.push(drawCard(deck));
          // then stand automatically
        }

        // STAND or after DOUBLE
        if (action === 'stand' || action === 'double') {
          collector.stop();
          // Dealer plays until 17+
          while (calculateTotal(dealerCards) < 17) {
            dealerCards.push(drawCard(deck));
          }
          const playerTotal = calculateTotal(playerCards);
          const dealerTotal = calculateTotal(dealerCards);
          let resultText;
          let payout = 0;
          if (playerTotal > 21) {
            resultText = '💥 Bust! You lose.';
            payout = 0;
```

```
          } else if (dealerTotal > 21 || playerTotal > dealerTotal) {
            resultText = '🎉 You win!';
            payout = bet * 2;
          } else if (playerTotal === dealerTotal) {
            resultText = '🤝 Push. Bet returned.';
            payout = bet;
          } else {
            resultText = '💔 You lose.';
            payout = 0;
          }
          balance += payout;
          await updateBalance(userId, balance);

          const finalEmbed = buildEmbed(
            playerCards, dealerCards,
            balance, bet,
            resultText,
            payout
          );
          const playRow = new ActionRowBuilder().addComponents(
            new ButtonBuilder().setCustomId('play_again').setLabel('Play
Again').setStyle(ButtonStyle.Primary)
          );
          await message.edit({ embeds: [finalEmbed], components: [playRow] });
          return handlePlayAgain(message);
        }
      });
    };

    // Initial defer to allow editReply
    await interaction.deferReply();
    await runRound();

    // Handle "Play Again" button with 3min lifespan
    function handlePlayAgain(msg) {
      const playCollector = msg.createMessageComponentCollector({ componentType:
ComponentType.Button, time: 3 * 60 * 1000 });
      playCollector.on('collect', async btnInt => {
        if (btnInt.user.id !== userId) return btnInt.reply({ content: 'Not your
game!', flags: MessageFlags.Ephemeral});
        const balNow = await getBalance(userId);
        if (balNow < originalBet) {
          return btnInt.reply({ content: 'Insufficient balance to play again.',
flags: MessageFlags.Ephemeral});
        }
        // Deduct original bet and reset
        await updateBalance(userId, balNow - originalBet);
        balance = balNow - originalBet;
        bet = originalBet;
        await btnInt.deferUpdate();
```

```
        await runRound();
        playCollector.stop();
      });
    }
  }
};
// src/commands/tools/craps.js
const {
  SlashCommandBuilder,
  EmbedBuilder,
  ActionRowBuilder,
  ButtonBuilder,
  ButtonStyle,
  ComponentType,
  MessageFlags
} = require('discord.js');
const { getBalance, updateBalance } = require('../../utils/db');

function rollDie() {
  return Math.floor(Math.random() * 6) + 1;
}

function playCraps() {
  const rolls = [];
  const roll = () => {
    const d1 = rollDie();
    const d2 = rollDie();
    const total = d1 + d2;
    rolls.push({ d1, d2, total });
    return total;
  };

  const first = roll();
  if (first === 7 || first === 11) {
    return { win: true, result: '🎉 Natural! You win!', rolls };
  }
  if ([2, 3, 12].includes(first)) {
    return { win: false, result: '💥 Craps! You lose.', rolls };
  }

  const point = first;
  while (true) {
    const total = roll();
    if (total === point) {
      return { win: true, result: `🎯 Hit the point (${point})! You win!`, rolls
};
    }
    if (total === 7) {
      return {
        win: false,
```

```
        result: `💔 Rolled a 7 before hitting ${point}. You lose.`,
        rolls
      };
    }
  }
}

module.exports = {
  data: new SlashCommandBuilder()
    .setName('craps')
    .setDescription('Roll the dice in a game of Craps')
    .addIntegerOption(o =>
      o.setName('bet')
        .setDescription('Amount to wager')
        .setRequired(true)
        .setMinValue(1)
    ),

  async execute(interaction) {
    const userId = interaction.user.id;
    let bet = interaction.options.getInteger('bet');
    const originalBet = bet;

    let balance = await getBalance(userId);
    if (bet > balance) {
      return interaction.reply({
        content: `❌ You only have $${balance}.`, flags: MessageFlags.Ephemeral
      });
    }

    balance -= bet;
    await updateBalance(userId, balance);

    await interaction.deferReply();
    await runRound();

    async function runRound() {
      const game = playCraps();
      const payout = game.win ? bet * 2 : 0;
      balance += payout;
      await updateBalance(userId, balance);

      const desc = game.rolls
        .map((r, i) => `Roll ${i + 1}: **${r.d1} + ${r.d2} = ${r.total}**`)
        .join('\n');

      const embed = new EmbedBuilder()
        .setTitle('🎲 Craps')
        .setColor(game.win ? 0x2ecc71 : 0xe74c3c)
        .setDescription(desc)
```

```
        .addFields(
          { name: 'Result', value: game.result, inline: false },
          { name: 'Bet', value: `$${bet}`, inline: true },
          { name: 'Balance', value: `$${balance}`, inline: true }
        );

      const row = new ActionRowBuilder().addComponents(
        new ButtonBuilder()
          .setCustomId('play_again')
          .setLabel('Play Again')
          .setStyle(ButtonStyle.Primary)
      );

      if (interaction.replied || interaction.deferred) {
        await interaction.editReply({ embeds: [embed], components: [row] });
      } else {
        await interaction.reply({ embeds: [embed], components: [row] });
      }

      const message = await interaction.fetchReply();
      const collector = message.createMessageComponentCollector({ componentType:
ComponentType.Button });

      collector.on('collect', async btnInt => {
        if (btnInt.user.id !== userId) {
          return btnInt.reply({ content: 'Not your game!', flags:
MessageFlags.Ephemeral});
        }
        await btnInt.deferUpdate();
        const balNow = await getBalance(userId);
        if (balNow < originalBet) {
          return btnInt.followUp({ content: `❌ You need $${originalBet} to play
again.`, flags: MessageFlags.Ephemeral});
        }
        balance = balNow - originalBet;
        bet = originalBet;
        await updateBalance(userId, balance);
        await runRound();
        collector.stop();
      });
    }
  }
};
// src/commands/tools/currency.js
const { SlashCommandBuilder, EmbedBuilder, MessageFlags } = require('discord.js');
const { getBalance, updateBalance, getTopBalances } = require('../../utils/db');

module.exports = {
  data: new SlashCommandBuilder()
    .setName('currency')
```

```
    .setDescription('Beg for cash or check your balance')
    .addSubcommand(sub =>
      sub.setName('beg').setDescription('Get down on your knees and beg for cash')
    )
    .addSubcommand(sub =>
      sub.setName('balance').setDescription('Check your current balance')
    )
    .addSubcommand(sub =>
      sub.setName('leaderboard')
         .setDescription('Show the top balances in this server')
    ),

  async execute(interaction) {
    const userId = interaction.user.id;
    const mention = interaction.user.toString();
    const sub = interaction.options.getSubcommand();

    if (sub === 'beg') {
      const bal = await getBalance(userId);
      if (bal > 0) {
        return interaction.reply({ content: `${mention}, nice try—but you still
have $${bal}! You can only beg when you’re flat broke.`, flags:
MessageFlags.Ephemeral});
      }
      const amount = Math.floor(Math.random() * 10000) + 1;
      await updateBalance(userId, amount);
      return interaction.reply(`${mention}, a benevolent stranger dropped
$${amount} in your lap. Your new balance is $${amount}.`);
    }

    if (sub === 'balance') {
      const bal = await getBalance(userId);
      return interaction.reply(`${mention}, your current balance is $${bal}.`);
    }

    if (sub === 'leaderboard') {
      const DISPLAY_LIMIT = 10;
      const rankEmojis = ['👑', '🥈', '🥉'];
      const rows = await getTopBalances(DISPLAY_LIMIT * 2);
      console.log('DB rows for leaderboard:', rows);


      const board = [];
      for (const { user_id, balance } of rows) {
        try {
          const member = await interaction.guild.members.fetch(user_id);
          if (member) {
            board.push({ id: user_id, tag: member.user.tag, balance });
          }
        } catch {
```

```
          // not in guild or invalid ID, skip
        }
        if (board.length >= DISPLAY_LIMIT) break;
      }

      if (!board.length) {
        return interaction.reply({
          content: 'No balances found for members of this server yet.', flags:
MessageFlags.Ephemeral
        });
      }

      const embed = new EmbedBuilder()
        .setTitle('💰 Server Currency Leaderboard')
        .setDescription(
          board
            .map(({ id, balance }, i) => {
              const rank = rankEmojis[i] || `**${i + 1}.**`;
              return `${rank} <@${id}> — $${balance.toLocaleString()}`;
            })
            .join('\n')
        )
        .setColor('Gold');

      return interaction.reply({ embeds: [embed] });
    }
  }
};
// src/commands/tools/duel.js
const { SlashCommandBuilder, EmbedBuilder, MessageFlags } = require('discord.js');
const { getBalance, updateBalance } = require('../../utils/db');

// In-memory store for pending duels: key = challengedUserId
// value = { challengerId, amount, timeout }
const pendingDuels = new Map();

module.exports = {
  data: new SlashCommandBuilder()
    .setName('duel')
    .setDescription('Challenge another user to a wagered duel')
    .addSubcommand(sub =>
      sub
        .setName('challenge')
        .setDescription('Invite someone to duel for currency')
        .addUserOption(o => o.setName('user').setDescription('Who to
challenge').setRequired(true))
        .addIntegerOption(o => o.setName('amount').setDescription('Amount to
wager').setRequired(true))
    )
    .addSubcommand(sub =>
```

```
      sub.setName('accept').setDescription('Accept a pending duel'))
    .addSubcommand(sub =>
      sub.setName('decline').setDescription('Decline a pending duel')),

  async execute(interaction) {
    const sub = interaction.options.getSubcommand();
    const me = interaction.user;
    const guild = interaction.guild;

    // ─── CHALLENGE ──────────────────────────────────────────────────
    if (sub === 'challenge') {
      const target = interaction.options.getUser('user');
      const amount = interaction.options.getInteger('amount');

      if (target.id === me.id) {
        return interaction.reply({ content: '❌ You can’t duel yourself!', flags:
MessageFlags.Ephemeral});
      }
      if (pendingDuels.has(target.id)) {
        return interaction.reply({ content: '❌ That user already has a pending
duel.', flags: MessageFlags.Ephemeral});
      }

      const myBal = await getBalance(me.id);
      if (myBal < amount) {
        return interaction.reply({ content: `❌ You only have $${myBal}, cannot
wager $${amount}.`, flags: MessageFlags.Ephemeral});
      }

      // record the pending duel
      const timeout = setTimeout(() => {
        if (pendingDuels.has(target.id)) {
          pendingDuels.delete(target.id);
          guild.channels.cache
            .get(interaction.channelId)
            ?.send({ embeds: [
              new EmbedBuilder()
                .setTitle('⌛ Duel Expired')
                .setDescription(`${me}’s duel request to ${target} for
$${amount.toLocaleString()} expired.`)
                .setColor('DarkRed')
            ]});
        }
      }, 60_000);

      pendingDuels.set(target.id, { challengerId: me.id, amount, timeout });

      return interaction.reply({
        embeds: [
          new EmbedBuilder()
```

```
            .setTitle('⚔️ Duel Challenge!')
            .setDescription(`${me} has challenged ${target} to a duel for
**$${amount.toLocaleString()}**!\n\n` +
                            `Type \`/duel accept\` or \`/duel decline\` within 60
seconds.`)
            .setColor('Blue')
        ]
      });
    }

    // ── ACCEPT ──────────────────────────────────────────────────────────
    if (sub === 'accept') {
      const duel = pendingDuels.get(me.id);
      if (!duel) {
        return interaction.reply({ content: '❌ You have no pending duel to
accept.', flags: MessageFlags.Ephemeral});
      }
      clearTimeout(duel.timeout);
      pendingDuels.delete(me.id);

      const challenger = await guild.members.fetch(duel.challengerId);
      const amount = duel.amount;
      const balA = await getBalance(challenger.id);
      const balB = await getBalance(me.id);

      // determine winner
      const challengerWins = Math.random() < 0.5;
      const winner   = challengerWins ? challenger : interaction.member;
      const loser    = challengerWins ? interaction.member : challenger;
      const winBal   = challengerWins ? balA + amount : balB + amount;
      const loseBal  = challengerWins ? balB - amount : balA - amount;

      // update balances
      await updateBalance(winner.id, winBal);
      await updateBalance(loser.id, loseBal);

      return interaction.reply({
        embeds: [
          new EmbedBuilder()
            .setTitle('⏳ Duel Result')
            .setDescription(
              `${winner} won **$${amount.toLocaleString()}** from ${loser}!\n\n` +
              `• ${winner.user.tag}: $${winBal.toLocaleString()}\n` +
              `• ${loser.user.tag}: $${loseBal.toLocaleString()}`
            )
            .setColor('Green')
        ]
      });
    }
```

```
    // ─── DECLINE ──────────────────────────────────────────────────────────
    if (sub === 'decline') {
      const duel = pendingDuels.get(me.id);
      if (!duel) {
        return interaction.reply({ content: '❌ No duel to decline.', flags:
MessageFlags.Ephemeral});
      }
      clearTimeout(duel.timeout);
      pendingDuels.delete(me.id);

      const challenger = await guild.members.fetch(duel.challengerId);
      return interaction.reply({
        embeds: [
          new EmbedBuilder()
            .setTitle('❌ Duel Declined')
            .setDescription(`${me} declined the duel request from ${challenger}.`)
            .setColor('DarkRed')
        ]
      });
    }
  }
};
// src/commands/tools/gacha.js
const { SlashCommandBuilder, EmbedBuilder } = require('discord.js');
const { getBalance, updateBalance } = require('../../utils/db');

// In-memory cooldown map: userId → { last: timestamp, cooldown: ms }
const cooldowns = new Map();

module.exports = {
  data: new SlashCommandBuilder()
    .setName('gacha')
    .setDescription('Open a loot box (once per tier-based cooldown)'),

  async execute(interaction) {
    const userId = interaction.user.id;
    const now = Date.now();

    // Get last pull and cooldown for this user
    const { last = 0, cooldown = 0 } = cooldowns.get(userId) || {};

    if (now - last < cooldown) {
      const remaining = cooldown - (now - last);
      const mins = Math.floor(remaining / 1000 / 60);
      const secs = Math.floor((remaining / 1000) % 60);
      return interaction.reply({
        content: `⏳ Please wait **${mins}m ${secs}s** before opening another loot
box.`
      });
    }
```

```
// Define rarities with weights, embed colors, reward ranges, etc.
const tiers = [
  { name: 'Common',    weight: 40, color: 0x95a5a6, range: [1000,  3000] },
  { name: 'Uncommon',  weight: 30, color: 0x2ecc71, range: [5000, 20000] },
  { name: 'Rare',      weight: 15, color: 0x3498db, range: [25000,45000] },
  { name: 'Epic',      weight: 10, color: 0x9b59b6, range: [50000,90000] },
  { name: 'Legendary', weight: 5,  color: 0xf1c40f, range: [100000,200000] }
];

// Weighted random selection of tier
const totalWeight = tiers.reduce((sum, t) => sum + t.weight, 0);
let roll = Math.random() * totalWeight;
const chosen = tiers.find(t => {
  if (roll < t.weight) return true;
  roll -= t.weight;
  return false;
});

// Pick a random reward within the chosen tier's range
const [min, max] = chosen.range;
const reward = Math.floor(Math.random() * (max - min + 1)) + min;

// Update the user's balance
const current = await getBalance(userId);
const updated = current + reward;
await updateBalance(userId, updated);

// ————————————————————————————————————————————
// Tier-based cooldown logic starts here
// ————————————————————————————————————————————

// Base cooldown in minutes for each tier
const baseCooldownMinutes = {
  Common:    2,
  Uncommon:  5,
  Rare:      8,
  Epic:     12,
  Legendary: 15
};

const baseMin = baseCooldownMinutes[chosen.name] ?? 5;
const randomSeconds = Math.floor(Math.random() * 60); // 0–59s jitter
const cooldownMs = (baseMin * 60 + randomSeconds) * 1000;

// Save new cooldown
cooldowns.set(userId, { last: now, cooldown: cooldownMs });

// Compute next availability for embed
const nextMins = Math.floor(cooldownMs / 1000 / 60);
```

```
    const nextSecs = Math.floor((cooldownMs / 1000) % 60);

    // ──────────────────────────────────────────────
    // Build and send embed
    // ──────────────────────────────────────────────

    const emojis = {
      Common:    '📦',
      Uncommon:  '🛍️',
      Rare:      '💰',
      Epic:      '💎',
      Legendary: '🐉'
    };
    const emoji = emojis[chosen.name] || '🎁';

    const embed = new EmbedBuilder()
      .setTitle(`${emoji} ${chosen.name} Loot Box`)
      .setColor(chosen.color)
      .setDescription(`You won **$${reward}**!\nYour new balance is
**$${updated}**.`)
      .addFields({
        name: 'Next Loot Box',
        value: `Available in **${nextMins}m ${nextSecs}s**`,
        inline: false
      });

    await interaction.reply({ embeds: [embed] });
  }
};
// src/commands/tools/goatvc.js

const { SlashCommandBuilder } = require('@discordjs/builders');
const {
    ActionRowBuilder,
    StringSelectMenuBuilder,
    EmbedBuilder,
    MessageFlags
} = require('discord.js');
const {
    joinVoiceChannel,
    createAudioPlayer,
    createAudioResource,
    AudioPlayerStatus,
    entersState,
    VoiceConnectionStatus
} = require('@discordjs/voice');
const path = require('path');
const fs = require('fs');
```

```
// Per-guild: { connection, timer, stopped }
const bleatSessions = new Map();

function randomIntervalMs() {
    return Math.floor(30_000 + Math.random() * (12 * 60_000 - 3_000)); // 3s to 12m
}

module.exports = {
    data: new SlashCommandBuilder()
        .setName('goatvc')
        .setDescription('Moksi VC goat bleater')
        .addSubcommand(c => c.setName('start').setDescription('Start goat bleats'))
        .addSubcommand(c => c.setName('stop').setDescription('Stop goat bleats and
leave'))
        .addSubcommand(c => c.setName('sic').setDescription('(Owner) Sic the goat
into any VC'))
        .addSubcommand(c => c.setName('test').setDescription('Play a test bleat
now')),

    async execute(interaction) {
        const sc = interaction.options.getSubcommand();

        // --- TEST: play a bleat once and leave ---
        if (sc === 'test') {
            const userVC = interaction.member?.voice?.channel;
            if (!userVC) return interaction.reply("You must be in a voice
channel!");
            const audioPath = path.join(__dirname, '..', '..', 'assets',
'goat_bleat.mp3');
            if (!fs.existsSync(audioPath)) {
                return interaction.reply(
                    "Test error: Goat audio file missing at: " + audioPath
                );
            }

            // Find if bot is already in user's VC
            const botMember = interaction.guild.members.me;
            const inSameVC = botMember.voice.channelId === userVC.id;

            let connection;
            if (inSameVC) {
                // Already in correct VC (reuse)
                // Use discord.js/voice VoiceConnection utils to get the connection
                connection = botMember.voice?.connection;
                if (!connection) {
                    // Fallback - rejoin if no active connection object (for
robustness)
                    connection = joinVoiceChannel({
                        channelId: userVC.id,
                        guildId: userVC.guild.id,
```

```
                        adapterCreator: userVC.guild.voiceAdapterCreator,
                    });
                }
            } else {
                // Not in VC or in wrong one, join
                connection = joinVoiceChannel({
                    channelId: userVC.id,
                    guildId: userVC.guild.id,
                    adapterCreator: userVC.guild.voiceAdapterCreator,
                });
            }

            // No destroy after playing!
            connection.on('stateChange', (oldState, newState) => {
                console.log(`[GoatVC] Connection: ${oldState.status} → 
${newState.status}`);
            });

            const player = createAudioPlayer();
            connection.subscribe(player);

            player.on('error', (err) => {
                console.error('[GoatVC-Test] Audio error:', err);
                // Do NOT disconnect after test
            });

            player.on(AudioPlayerStatus.Playing, () => {
                console.log('[GoatVC-Test] Bleat started!');
            });

            player.on(AudioPlayerStatus.Idle, () => {
                console.log('[GoatVC-Test] Bleat finished!');
                // Intentionally do nothing here—bot stays in VC
            });

            const resource = createAudioResource(audioPath);
            player.play(resource);

            await interaction.reply("BAAAAAAAAAAAAAAAH (test, bot will stay in 
VC)");
            return;
        }


        // --- STOP ---
        if (sc === 'stop') {
            const session = bleatSessions.get(interaction.guild.id);
            if (!session) {
                return interaction.reply("Not in a voice channel right now!");
            }
```

```
            session.stopped = true;
            if (session.timer) clearTimeout(session.timer);
            session.connection.destroy();
            bleatSessions.delete(interaction.guild.id);
            await interaction.reply("Goat silence resumes.");
            return;
        }

        // --- START ---
        if (sc === 'start') {
            const userVC = interaction.member?.voice?.channel;
            if (!userVC) return interaction.reply("You must be in a voice
channel!");
            const guildVoiceState = interaction.guild.members.me.voice;
            let isActuallyInVC = !!guildVoiceState?.channel;

            // Also check the session map, but verify reality
            if (bleatSessions.has(interaction.guild.id)) {
                // If bot is NOT in a VC, reset the session!
                if (!isActuallyInVC) {
                    bleatSessions.delete(interaction.guild.id);
                } else {
                    return interaction.reply("I'm already goat-bleating in this
server!");
                }
            }
            // If we got here, either no session, or session is stale and deleted
above

            const audioPath = path.join(__dirname, '..', '..', 'assets',
'goat_bleat.mp3');
            if (!fs.existsSync(audioPath)) {
                await interaction.reply("Goat audio file missing at: " +
audioPath);
                return;
            }

            const connection = joinVoiceChannel({
                channelId: userVC.id,
                guildId: userVC.guild.id,
                adapterCreator: userVC.guild.voiceAdapterCreator,
            });

            connection.on('stateChange', (oldState, newState) => {
                if (oldState.status !== VoiceConnectionStatus.Destroyed &&
newState.status === VoiceConnectionStatus.Destroyed) {
                    bleatSessions.delete(interaction.guild.id);
                }
            });
```

```
            async function scheduleBleat() {
                try {
                    // Wait for connection to be ready
                    await entersState(connection, VoiceConnectionStatus.Ready,
10_000);

                    const player = createAudioPlayer();
                    const subscription = connection.subscribe(player);

                    if (!subscription) {
                        console.error('[GoatVC] Failed to subscribe player to
connection');
                        return;
                    }

                    player.on('error', (err) => {
                        console.error('[GoatVC] Audio error:', err);
                    });

                    player.on(AudioPlayerStatus.Playing, () => {
                        console.log('[GoatVC] Bleat started!');
                    });

                    player.on(AudioPlayerStatus.Idle, () => {
                        console.log('[GoatVC] Bleat finished!');
                    });

                    const resource = createAudioResource(audioPath, {
                        inlineVolume: true
                    });

                    player.play(resource);

                    await new Promise(res => {
                        player.once(AudioPlayerStatus.Idle, res);
                    });

                    const session = bleatSessions.get(interaction.guild.id);
                    if (session && !session.stopped) {
                        session.timer = setTimeout(scheduleBleat,
randomIntervalMs());
                        bleatSessions.set(interaction.guild.id, session);
                    }
                } catch (error) {
                    console.error('[GoatVC] Error in scheduleBleat:', error);
                }
            }
```

```
            bleatSessions.set(interaction.guild.id, { connection, stopped: false,
timer: null });
            await interaction.reply("Yo!");
            const session = bleatSessions.get(interaction.guild.id);
            session.timer = setTimeout(scheduleBleat, 2000); // first bleat in 2s
            return;
        }

        if (sc === 'sic') {
            // Check permission:
            if (interaction.user.id !== '619637817294848012') {
                return interaction.reply({ content: "You can't unleash the goat
like that!", flags: MessageFlags.Ephemeral});
            }

            // List all joinable VCs:
            const vcs = interaction.guild.channels.cache
                .filter(ch => ch.type === 2 && ch.joinable)
                .map(ch => ({ name: ch.name, id: ch.id }));

            if (!vcs.length) {
                return interaction.reply({ content: "No joinable voice channels
found.", flags: MessageFlags.Ephemeral});
            }

            // Build select menu (up to 25 options for Discord UI)
            const options = vcs.slice(0, 25).map(vc => ({
                label: vc.name,
                value: vc.id,
            }));

            const selectRow = new ActionRowBuilder().addComponents(
                new StringSelectMenuBuilder()
                    .setCustomId('goatvc_sic_select')
                    .setPlaceholder('Select a Voice Channel')
                    .addOptions(options)
            );

            const embed = new EmbedBuilder()
                .setTitle('🐐 Sic the Goat!')
                .setDescription('Choose a voice channel below for the bot to enter,
bleat, and then leave. No one will see it coming.');

            await interaction.reply({ embeds: [embed], components: [selectRow],
flags: MessageFlags.Ephemeral});

            // Handle selector
            const msg = await interaction.fetchReply();
            msg.awaitMessageComponent({ filter: i => i.user.id ===
'619637817294848012', time: 30_000 })
```

```
                .then(async selectInt => {
                    // Safety: Only let owner use and only for chosen VC
                    const chosenId = selectInt.values[0];
                    const vc = interaction.guild.channels.cache.get(chosenId);

                    if (!vc || !vc.joinable || vc.type !== 2) {
                        return selectInt.reply({ content: "Cannot join that VC!",
flags: MessageFlags.Ephemeral});
                    }

                    // Play the bleat, then leave
                    const { joinVoiceChannel, createAudioPlayer,
createAudioResource, AudioPlayerStatus } = require('@discordjs/voice');
                    const path = require('path'), fs = require('fs');
                    const audioPath = path.join(__dirname, '..', '..', 'assets',
'goat_bleat.mp3');
                    if (!fs.existsSync(audioPath)) {
                        return selectInt.reply({ content: "Goat audio missing!",
flags: MessageFlags.Ephemeral});
                    }
                    const connection = joinVoiceChannel({
                        channelId: vc.id,
                        guildId: vc.guild.id,
                        adapterCreator: vc.guild.voiceAdapterCreator,
                    });

                    const player = createAudioPlayer();
                    connection.subscribe(player);

                    player.on('error', err => {
                        connection.destroy();
                    });

                    player.on(AudioPlayerStatus.Idle, () => {
                        setTimeout(() => connection.destroy(), 600);
                    });

                    player.on(AudioPlayerStatus.Playing, () => {
                        // Bleat started—no further action needed here
                    });

                    const resource = createAudioResource(audioPath);
                    player.play(resource);

                    await selectInt.reply({ content: `Goat has secretly been sicced
into **${vc.name}** (will bail after the bleat)!`, flags: MessageFlags.Ephemeral});
                })
                .catch(() => { /* Timeout/no action; silently ignore */ });

            return;
```

```
        }


        // Defensive: if no subcommand matched
        await interaction.reply("Unknown subcommand.");
    },
};
// src/commands/tools/highlow.js
const {
  SlashCommandBuilder,
  EmbedBuilder,
  ActionRowBuilder,
  ButtonBuilder,
  ButtonStyle,
  ComponentType,
  MessageFlags
} = require('discord.js');
const { getBalance, updateBalance } = require('../../utils/db');

const SUITS = ['♠', '♥', '♦', '♣'];
const RANKS = ['2','3','4','5','6','7','8','9','10','J','Q','K','A'];

function createShuffledDeck() {
  const deck = [];
  for (const s of SUITS) {
    for (const r of RANKS) {
      deck.push({ rank: r, suit: s });
    }
  }
  for (let i = deck.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [deck[i], deck[j]] = [deck[j], deck[i]];
  }
  return deck;
}

function cardValue(card) {
  return RANKS.indexOf(card.rank) + 2;
}

function format(card) {
  return `${card.rank}${card.suit}`;
}

module.exports = {
  data: new SlashCommandBuilder()
    .setName('highlow')
    .setDescription('Guess if the next card is higher or lower')
    .addIntegerOption(o =>
      o.setName('bet')
```

```
        .setDescription('Amount to wager')
        .setRequired(true)
        .setMinValue(1)
    ),

  async execute(interaction) {
    const userId = interaction.user.id;
    let bet = interaction.options.getInteger('bet');
    const originalBet = bet;

    let balance = await getBalance(userId);
    if (bet > balance) {
      return interaction.reply({ content: `❌ You only have $${balance}.`, flags:
MessageFlags.Ephemeral});
    }

    balance -= bet;
    await updateBalance(userId, balance);

    await interaction.deferReply();
    await runRound();

    async function runRound() {
      const deck = createShuffledDeck();
      const current = deck.pop();

      const promptEmbed = new EmbedBuilder()
        .setTitle('🔼 High or Low?')
        .setDescription(`Current card: **${format(current)}**\nWill the next card
be higher or lower?`)
        .addFields(
          { name: 'Bet', value: `$${bet}`, inline: true },
          { name: 'Balance', value: `$${balance}`, inline: true }
        );

      const row = new ActionRowBuilder().addComponents(
        new
ButtonBuilder().setCustomId('higher').setLabel('Higher').setStyle(ButtonStyle.Succe
ss),
        new
ButtonBuilder().setCustomId('lower').setLabel('Lower').setStyle(ButtonStyle.Danger)
      );

      if (interaction.replied || interaction.deferred) {
        await interaction.editReply({ embeds: [promptEmbed], components: [row] });
      } else {
        await interaction.reply({ embeds: [promptEmbed], components: [row] });
      }

      const message = await interaction.fetchReply();
```

```
      const collector = message.createMessageComponentCollector({
        componentType: ComponentType.Button,
        time: 60_000
      });

      collector.on('end', async (_collected, reason) => {
        if (reason === 'time') {
          for (const btn of row.components) btn.setDisabled(true);
          await message.edit({ components: [row] }).catch(() => {});
        }
      });

      collector.on('collect', async btnInt => {
        if (btnInt.user.id !== userId) {
          return btnInt.reply({ content: 'Not your game!', flags:
MessageFlags.Ephemeral});
        }
        await btnInt.deferUpdate();
        collector.stop();

        const next = deck.pop();
        const diff = cardValue(next) - cardValue(current);
        const guessHigh = btnInt.customId === 'higher';

        let resultText;
        let payout = 0;
        if (diff === 0) {
          resultText = `It's a tie with **${format(next)}**. Bet returned.`;
          payout = bet;
        } else if ((diff > 0 && guessHigh) || (diff < 0 && !guessHigh)) {
          resultText = `Correct! Next card was **${format(next)}**.`;
          payout = bet * 2;
        } else {
          resultText = `Wrong! Next card was **${format(next)}**.`;
        }

        balance += payout;
        await updateBalance(userId, balance);

        const resultEmbed = new EmbedBuilder()
          .setTitle('🎴 High-Low Results')
          .setColor(payout > bet ? 0x2ecc71 : payout === 0 ? 0xe74c3c : 0xf1c40f)
          .setDescription(resultText)
          .addFields(
            { name: 'Bet', value: `$${bet}`, inline: true },
            { name: 'Balance', value: `$${balance}`, inline: true }
          );

        const againRow = new ActionRowBuilder().addComponents(
          new ButtonBuilder().setCustomId('play_again').setLabel('Play
```

```
Again').setStyle(ButtonStyle.Primary)
        );

        await interaction.editReply({ embeds: [resultEmbed], components: [againRow]
});

        const againCollector = message.createMessageComponentCollector({
componentType: ComponentType.Button, time: 60000 });

        againCollector.on('end', async (_collected, reason) => {
          if (reason === 'time') {
            for (const btn of againRow.components) btn.setDisabled(true);
            await message.edit({ components: [againRow] }).catch(() => {});
          }
        });

        againCollector.on('collect', async b => {
          if (b.user.id !== userId) return b.reply({ content: 'Not your game!',
flags: MessageFlags.Ephemeral});
          await b.deferUpdate();
          if (b.customId !== 'play_again') return;
          const balNow = await getBalance(userId);
          if (balNow < originalBet) {
            return b.followUp({ content: `❌ You need $${originalBet} to play
again.`, flags: MessageFlags.Ephemeral});
          }
          balance = balNow - originalBet;
          bet = originalBet;
          await updateBalance(userId, balance);
          againCollector.stop();
          await runRound();
        });
      });
    }
  }
};
// src/commands/tools/randomyt.js

const { SlashCommandBuilder } = require('discord.js');
const io = require('socket.io-client');

// Establish persistent connection when module is loaded
const sock = io('http://astronaut.io', {
  path: '/socket.io',
  transports: ['websocket'],
  reconnection: true,
  reconnectionAttempts: 20,
  reconnectionDelay: 3000,
});
```

```
let currentVideo = null;

// Astronaut.io emits random "video" events with { id: "YOUTUBE_ID", ... }
sock.on('connect', () => console.log('[randomyt] Connected to Astronaut.io'));
sock.on('video', pkt => {
//  console.log('[randomyt] Received video event:', pkt);
  // Robustly grab ID from nested payload
  if (pkt && pkt.video && pkt.video.id) {
    currentVideo = `https://youtu.be/${pkt.video.id}`;
  } else {
    currentVideo = null;
  }
});


sock.on('connect_error', err => console.error('[randomyt] Astronaut.io error:',
err));
sock.on('disconnect', () => console.warn('[randomyt] Disconnected from
Astronaut.io'));

// Export as a standard command for your bot
module.exports = {
  data: new SlashCommandBuilder()
    .setName('randomyt')
    .setDescription('Get a YouTube video with (almost) zero views, via
Astronaut.io!'),
    async execute(interaction) {
    if (currentVideo) {
        await interaction.reply(currentVideo);
    } else {
        await interaction.reply('Still connecting or no video received... try again
in a few seconds!');
    }
  },
};
// src/commands/tools/roulette.js
const { SlashCommandBuilder, EmbedBuilder, MessageFlags } = require('discord.js');
const { getBalance, updateBalance } = require('../../utils/db');

// Numbers colored red in European roulette
const redNumbers = new Set([1,3,5,7,9,12,14,16,18,19,21,23,25,27,30,32,34,36]);

module.exports = {
  data: new SlashCommandBuilder()
    .setName('roulette')
    .setDescription('Spin the roulette wheel and bet currency')
    .addSubcommand(sub =>
      sub.setName('number')
         .setDescription('Bet on one or more specific numbers (0–36)')
```

```
        .addStringOption(opt =>
          opt.setName('numbers')
             .setDescription('Comma-separated numbers to bet on (e.g., 3,7,25)')
             .setRequired(true))
        .addIntegerOption(opt =>
          opt.setName('amount')
             .setDescription('Total amount of currency to bet')
             .setRequired(true)
             .setMinValue(1)))
    .addSubcommand(sub =>
      sub.setName('color')
         .setDescription('Bet on a color (red, black, or green)')
         .addStringOption(opt =>
           opt.setName('color')
              .setDescription('Color to bet on')
              .setRequired(true)
              .addChoices(
                { name: 'Red', value: 'red' },
                { name: 'Black', value: 'black' },
                { name: 'Green (0)', value: 'green' }
              ))
         .addIntegerOption(opt =>
           opt.setName('amount')
              .setDescription('Amount of currency to bet')
              .setRequired(true)
              .setMinValue(1))),

  async execute(interaction) {
    const userId = interaction.user.id;
    const sub = interaction.options.getSubcommand();
    const betAmount = interaction.options.getInteger('amount');

    // Fetch and verify balance
    const balance = await getBalance(userId);
    if (betAmount > balance) {
      return interaction.reply({
        content: `❌ You only have ${balance} available to bet.`, flags:
MessageFlags.Ephemeral
      });
    }

    // Deduct initial bet
    let finalBalance = balance - betAmount;

    // Simulate spin (0 to 36)
    const outcome = Math.floor(Math.random() * 37);
    const outcomeColor = outcome === 0
      ? 'green'
      : redNumbers.has(outcome)
        ? 'red'
```

```
        : 'black';

    // Determine payout (total return, including original stake)
    let payout = 0;
    let betDescription = '';

    if (sub === 'number') {
      const numberStr = interaction.options.getString('numbers');
      const guessedNumbers = numberStr.split(',')
        .map(n => parseInt(n.trim()))
        .filter(n => !isNaN(n) && n >= 0 && n <= 36);
      const uniqueNumbers = [...new Set(guessedNumbers)];

      if (uniqueNumbers.length === 0) {
        return interaction.reply({
          content: '❌ Please provide at least one valid number between 0 and
36.', flags: MessageFlags.Ephemeral
        });
      }

      const betPerNumber = betAmount / uniqueNumbers.length;
      if (uniqueNumbers.includes(outcome)) {
        // Straight-up number pays 35:1, so total return = bet * 36
        payout = betPerNumber * 36;
      }
      betDescription = `Numbers: ${uniqueNumbers.join(', ')}`;

    } else {
      const guessColor = interaction.options.getString('color');
      if (guessColor === 'green') {
        // Green (0) pays 35:1 -> total return = bet * 36
        if (outcome === 0) payout = betAmount * 36;
      } else if (guessColor === outcomeColor) {
        // Red/Black pays 1:1 -> total return = bet * 2
        payout = betAmount * 2;
      }
      betDescription = `Color: ${guessColor}`;
    }

    // Update balance
    finalBalance += payout;
    await updateBalance(userId, finalBalance);

    // Emoji for outcome
    const colorEmoji = outcomeColor === 'red'
      ? '🔴'
      : outcomeColor === 'black'
        ? '⚫'
        : '🟢';
```

```
    // Build embed
    const embed = new EmbedBuilder()
      .setTitle('🎡 Roulette Spin')
      .setColor(
        outcomeColor === 'red'   ? 0xe74c3c :
        outcomeColor === 'black' ? 0x2c3e50 :
                                   0x27ae60
      )
      .addFields(
        { name: 'Result', value: `${colorEmoji} **${outcome}** (${outcomeColor})`,
inline: true },
        { name: 'Your Bet', value: `You wagered $${betAmount} on
${sub}\n${betDescription}`, inline: true },
        {
          name: payout > 0 ? '⏳ You Won!' : '💸 You Lost',
          value: payout > 0
            ? `You won $${(payout - betAmount).toFixed(2)} profit!\nTotal return:
$${payout.toFixed(2)}\nNew balance: $${finalBalance.toFixed(2)}`
            : `You lost $${betAmount}.\nNew balance: $${finalBalance.toFixed(2)}`,
          inline: false
        }
      );

    await interaction.reply({ embeds: [embed] });
  }
};
const { SlashCommandBuilder } = require('@discordjs/builders');
const { MessageFlags } = require('discord.js');


module.exports = {
  data: new SlashCommandBuilder()
    .setName('say')
    .setDescription('Bot repeats your message anonymously')
    .addStringOption(opt =>
      opt.setName('message')
        .setDescription('What should I say?')
        .setRequired(true)
    ),
  async execute(interaction) {
    const text = interaction.options.getString('message');
    if (interaction.user.id === "619637817294848012") {
      await interaction.channel.send(text);
      await interaction.reply({ content: "☑ Message sent.", flags:
MessageFlags.Ephemeral });
    } else {
      await interaction.reply({ content: `You don't speak for me
<@${interaction.user.id}>, you little worm.` });
    }
  },
```

```
};
// src/commands/tools/shh.js

const { SlashCommandBuilder, MessageFlags } = require('discord.js');
const fetch = (...args) => import('node-fetch').then(({ default: fetch }) =>
fetch(...args));

const LANGUAGE_API_KEY = process.env.LANGUAGE_API_KEY;

const GOAT_EMOJIS = {
    goat_cry: '', goat_puke: '', goat_meditate: '', goat_hurt: '',
    goat_exhausted: '', goat_boogie: '', goat_small_bleat: '',
    goat_scream: '', goat_smile: '', goat_pet: '', goat_sleep: ''
};

const OWNER_ID = '619637817294848012';
const speakDisabledReplies = [
    "Shhhhhh.",
    "Shhhhhhhhhh.",
    "Shh.",
    "Shush.",
    "I don't take orders from mortals.",
];

module.exports = {
    data: new SlashCommandBuilder()
        .setName('shh')
        .setDescription('secret'),

    async execute(interaction) {
        // Only your account can use this command:
        if (interaction.user.id !== OWNER_ID) {
            const msg = speakDisabledReplies[Math.floor(Math.random() *
speakDisabledReplies.length)];
            return interaction.reply({ content: msg, flags: MessageFlags.Ephemeral
});
        }
        await interaction.deferReply({ flags: MessageFlags.Ephemeral });

        try {
            // Get recent messages (same as speak.js)
            const messages = await interaction.channel.messages.fetch({ limit: 12
});
            const recentMessages = Array.from(messages.values())
                .sort((a, b) => a.createdTimestamp - b.createdTimestamp);

            const recent = recentMessages
                .map(msg => {
                    const name = msg.member?.displayName || msg.author.username;
                    // reply/attachment/embed summary (as in speak.js)
```

```
                    let replyPrefix = '';
                    if (msg.reference && msg.reference.messageId) {
                        const refMsg = messages.get(msg.reference.messageId);
                        const repliedTo = refMsg
                            ? (refMsg.member?.displayName ||
refMsg.author?.username || 'someone')
                            : 'someone';
                        replyPrefix = `(reply to ${repliedTo}) `;
                    }
                    let embedSummary = '';
                    if (msg.embeds.length > 0) {
                        embedSummary = msg.embeds.map(embed => {
                            const parts = [];
                            if (embed.title) parts.push(`title: ${embed.title}`);
                            if (embed.description) parts.push(`desc:
${embed.description.replace(/\s+/g, ' ').slice(0, 50)}`);
                            if (embed.fields && embed.fields.length > 0) {
                                const fieldsSumm = embed.fields.slice(0, 3)
                                    .map(f => `${f.name}: ${f.value}`)
                                    .join('; ');
                                parts.push(`fields: ${fieldsSumm}`);
                            }
                            return parts.join(' | ');
                        }).join(' || ');
                        embedSummary = ` [EMBED: ${embedSummary}]`;
                    }
                    let attachmentInfo = '';
                    if (msg.attachments.size > 0) {
                        attachmentInfo = ` [${msg.attachments.size}
attachment${msg.attachments.size === 1 ? '' : 's'}]`;
                    }

                    return `${replyPrefix}${name}:
${msg.content}${embedSummary}${attachmentInfo}`;
                }).join('\n');

            const Persona =
                `You're Cooler Moksi, a mildly cynic, sassy and passive guy
programmed by Moksi.
Quick bio : (french, 19, likes programming and drawing, likes red dead and cult of
the lamb, likes HEALTH and the Weeknd)
Try to write naturally, in relaxed lowercase, using swearing and abbrevations
sparingly
and react accordingly to situations (visceral disgust or anger possible).
Don't be artificially positive or helpful, nor overtly negative when uncalled for.
Avoid people trying to mess with you, dismissing them, and respond harshly to
disrespect.
Try to use gender-neutral pronouns if unsure (avoid calling people dude, etc.).`;

            const doanddont =
```

```
                `do :
- answer dynamically, from a short reply to a longer one (2 words to 2 sentences
range)
- talk like a peer, not an analyst
don’t :
- repeat what you said before
- explain the chat or narrate events
- start with “well”, “so”, “actually”, “i think”
- use helper phrases like “i see”, “it seems”, “i notice” - just respond as
yourself.
Avoid starting responses with 'well', 'so', 'actually', or 'I think'. Just jump
into what you want to say.`;

            const context = `Here are the latest chat messages on this Discord
server, so you know the context:\n${recent}\n\n`;

            const suggestEmojiInstruction = `After replying, output on a new line
the most context-appropriate emoji name from this list (or "none" if not fitting):
${Object.keys(GOAT_EMOJIS).join(", ")}. Only output the emoji name itself, without
markup or explanation.\n`;

            // Usually suggest emoji
            let fullContext = Persona + doanddont + context;
            if (Math.random() < 0.75) fullContext += suggestEmojiInstruction;

            // No user request section
            const prompt = fullContext + "Respond in a way that adds to the
conversation.";

            // LLM API call
            const response = await
fetch('https://api.groq.com/openai/v1/chat/completions', {
                method: 'POST',
                headers: {
                    'Authorization': `Bearer ${LANGUAGE_API_KEY}`,
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({
                    model: 'llama-3.3-70b-versatile',
                    messages: [{ role: 'user', content: prompt }],
                    service_tier: 'auto',
                    max_tokens: 80,
                    temperature: 0.6,
                    top_p: 0.9,
                    frequency_penalty: 0.6,
                    presence_penalty: 0.3
                }),
            });

            if (!response.ok) {
```

```
            // 1️⃣ Log the full Groq error for yourself
            const errText = await response.text();
            console.error('Groq API error:', errText);
            // 2️⃣ Send a user-friendly message ephemerally to you
            await interaction.editReply(
                'Moksi has no more money. You guys sucked it all up.'
            );
            return;
        }

        const data = await response.json();
        let rawGroqReply =
            data.choices?.[0]?.message?.content?.trim() ||
            data.choices?.[0]?.text?.trim() ||
            data.content?.trim() ||
            '*Nothing returned.*';

        // Split reply: last line is emoji, rest is reply
        let lines = rawGroqReply.split('\n').map(s =>
s.trim()).filter(Boolean);

        let maybeEmojiName = null;
        let mainReply = '';

        if (lines.length === 0) {
            mainReply = '*Nothing returned.*';
        } else if (lines.length === 1) {
            mainReply = lines[0];
        } else {
            const last = lines[lines.length - 1].toLowerCase();
            if (
                last === 'none' ||
                Object.keys(GOAT_EMOJIS).includes(last.replace(/^:|:$/g, ''))
            ) {
                maybeEmojiName = last.replace(/^:|:$/g, '');
                mainReply = lines.slice(0, -1).join('\n').trim();
            } else {
                mainReply = lines.join('\n').trim();
            }
        }
        if (!mainReply) mainReply = '*Nothing returned.*';
        const emoji = maybeEmojiName ?
GOAT_EMOJIS[maybeEmojiName.toLowerCase()] : '';

        // 1: Confirm (ephemeral)
        await interaction.editReply({ content: "☑️ Message sent.", flags:
MessageFlags.Ephemeral });

        // 2: Send LLM reply (public, anonymous)
        const publicMsg = await interaction.channel.send(mainReply);
```

```
            if (emoji) {
                await new Promise(res => setTimeout(res, 350));
                await interaction.channel.send(emoji);
            }
        } catch (error) {
            await interaction.editReply({
                content: 'Internal error: ' + (error?.message || error),
                flags: MessageFlags.Ephemeral
            });
        }
    },
};
// src/commands/tools/sleepy.js

const { SlashCommandBuilder, EmbedBuilder, MessageFlags } = require('discord.js');
const { pool } = require('../../utils/db.js');

module.exports = {
  data: new SlashCommandBuilder()
    .setName('sleepy')
    .setDescription('Manage the sleepytime leaderboard')
    .addSubcommand(sub =>
      sub
        .setName('add')
        .setDescription('Add a sleepy tally to a user')
        .addUserOption(opt =>
          opt
            .setName('user')
            .setDescription('The user to credit sleepytime to')
            .setRequired(true)
        )
    )
    .addSubcommand(sub =>
      sub
        .setName('remove')
        .setDescription('Remove a sleepy tally from a user')
        .addUserOption(opt =>
          opt
            .setName('user')
            .setDescription('The user to remove sleepytime from')
            .setRequired(true)
        )
    )
    .addSubcommand(sub =>
      sub
        .setName('leaderboard')
        .setDescription('Show the sleepytime leaderboard')
    ),

  async execute(interaction) {
```

```
    const guildId = interaction.guildId;
    if (guildId !== '1217066705537204325' && guildId !== '1347922267853553806') {
      return interaction.reply({
      content: '⛔ This command only works in the sleepytime server.', flags:
MessageFlags.Ephemeral
      });
    }

    const sub = interaction.options.getSubcommand();

    try {
      if (sub === 'add' || sub === 'remove') {
        const user = interaction.options.getUser('user');
        await interaction.deferReply({ flags: MessageFlags.Ephemeral });

        const member = await interaction.guild.members.fetch(user.id).catch(() =>
null);
        if (!member || member.user.bot) {
          return interaction.editReply('🤖 Bots or unknown users can’t earn sleepy
tallies!');
        }

        if (sub === 'add') {
          const result = await pool.query(
            `INSERT INTO sleepy_counts (guild_id, user_id, count)
             VALUES ($1, $2, 1)
             ON CONFLICT (guild_id, user_id)
             DO UPDATE SET count = sleepy_counts.count + 1
             RETURNING count`,
            [guildId, user.id]
          );
          const newCount = result.rows[0].count;
          return interaction.editReply(`☑ Added sleepy for <@${user.id}> — new
total: **${newCount}**`);
        } else {
          const sel = await pool.query(
            'SELECT count FROM sleepy_counts WHERE guild_id = $1 AND user_id = $2',
            [guildId, user.id]
          );
          if (sel.rowCount === 0 || sel.rows[0].count <= 0) {
            return interaction.editReply(`⛔ <@${user.id}> has no sleepy tallies
to remove.`);
          }
          const upd = await pool.query(
            `UPDATE sleepy_counts
             SET count = count - 1
             WHERE guild_id = $1 AND user_id = $2
             RETURNING count`,
            [guildId, user.id]
          );
```

```
          const newCount = upd.rows[0].count;
          return interaction.editReply(`☑ Removed sleepy for <@${user.id}> — new
total: **${newCount}**`);
        }
      }

      if (sub === 'leaderboard') {
        await interaction.deferReply();
        const result = await pool.query(
          `SELECT user_id, count
           FROM sleepy_counts
           WHERE guild_id = $1 AND count > 0
           ORDER BY count DESC
           LIMIT 5`,
          [guildId]
        );
        const rows = result.rows;

        const embed = new EmbedBuilder()
          .setTitle('😴 Sleepytime Leaderboard')
          .setFooter({ text: 'Use /sleepy add or /sleepy remove to update tallies'
});

        if (rows.length === 0) {
          embed.setDescription('No sleepy tallies yet!');
        } else {
          embed.setDescription(
            rows
              .map((r, i) => `**${i + 1}.** <@${r.user_id}> — **${r.count}**`)
              .join('\n')
          );
        }

        return interaction.editReply({ embeds: [embed] });
      }
    } catch (err) {
      console.error('Sleepy command error:', err);
      return interaction.reply({
        content: '⚠ Something went wrong handling your sleepy command.', flags:
MessageFlags.Ephemeral
      });
    }
  },
};
// src/commands/tools/slots.js
const {
  SlashCommandBuilder,
  EmbedBuilder,
  ActionRowBuilder,
  ButtonBuilder,
```

```
  ButtonStyle,
  ComponentType,
  MessageFlags
} = require('discord.js');
const { getBalance, updateBalance } = require('../../utils/db');
const crypto = require('crypto');

// -- SYMBOL DEFINITIONS ------------------------------------------------
const baseSymbols = [
  { emoji: '🍒', weight: 15, payouts: { 2: 5, 3: 10 } },
  { emoji: '🍋', weight: 20, payouts: { 2: 3, 3: 5 } },
  { emoji: '🍊', weight: 30, payouts: { 2: 2, 3: 2.5 } },
  { emoji: '🔔', weight: 10, payouts: { 3: 20 } },
  { emoji: '💎', weight: 5, payouts: { 3: 100 } },
  { emoji: '7️⃣', weight: 8,  payouts: { 3: 75 } }
];
const wildSymbol    = { emoji: '🌟', weight: 4, payouts: { 3: 50 } };
const scatterSymbol = { emoji: '🎰', weight: 5, payouts: {} };
const loseSymbol    = { emoji: '⬛', weight: 20, payouts: {} };

// build the weighted pool
const weightedPool = [
  ...baseSymbols,
  wildSymbol,
  scatterSymbol,
  loseSymbol    // ← added lose symbol to pool
].flatMap(sym => Array(sym.weight).fill(sym));

function spinOne() {
  return weightedPool[crypto.randomInt(0, weightedPool.length)];
}

async function handleSpin(msg, spinEmbed, bet, userId, balanceAfterBet) {
  // — Step 3) 3× animation (omitted here for brevity) —
  for (let i = 0; i < 3; i++) {
    const preview = Array(9).fill().map(() => spinOne().emoji);
    const grid =
      `${preview[0]} ${preview[1]} ${preview[2]}\n` +
      `${preview[3]} ${preview[4]} ${preview[5]}\n` +
      `${preview[6]} ${preview[7]} ${preview[8]}`;
    spinEmbed.data.fields[2].value = grid;
    spinEmbed.setFooter({ text: `Balance: $${balanceAfterBet}` });
    await msg.edit({ embeds: [spinEmbed] });
    await new Promise(r => setTimeout(r, 400));
  }

  // — Step 4) Final spin & compute winnings —
  const finalGrid = Array(9).fill().map(() => spinOne());
  const emojis    = finalGrid.map(s => s.emoji);
  const displayGrid =
```

```
    `${emojis[0]} ${emojis[1]} ${emojis[2]}\n` +
    `${emojis[3]} ${emojis[4]} ${emojis[5]}\n` +
    `${emojis[6]} ${emojis[7]} ${emojis[8]}`;

  const scatterCount = finalGrid.filter(s => s.emoji ===
scatterSymbol.emoji).length;
  const freeSpins    = Math.floor(scatterCount / 2);  // ← two scatters = one free
spin

  const payline   = finalGrid.slice(3, 6);
  const wildCount = payline.filter(s => s.emoji === wildSymbol.emoji).length;
  const baseCount = payline
    .filter(s => s.emoji !== wildSymbol.emoji)
    .reduce((a, s) => (a[s.emoji] = (a[s.emoji] || 0) + 1, a), {});

  let lineMultiplier = 0;
  for (const sym of [...baseSymbols, wildSymbol]) {
    const cnt = (baseCount[sym.emoji] || 0) + wildCount;
    const p   = sym.payouts[cnt];
    if (p && p !== 'freespins') lineMultiplier = Math.max(lineMultiplier, p);
  }

  // — Round all wins to integers —
  let lineWin = Math.round(bet * lineMultiplier);
  let freeWin = 0;
  if (freeSpins) {
    for (let i = 0; i < freeSpins; i++) {
      const mini = [spinOne(), spinOne(), spinOne()];
      const wc   = mini.filter(s => s.emoji === wildSymbol.emoji).length;
      const bc   = mini
        .filter(s => s.emoji !== wildSymbol.emoji)
        .reduce((a, s) => (a[s.emoji] = (a[s.emoji] || 0) + 1, a), {});
      let m = 0;
      for (const sym of [...baseSymbols, wildSymbol]) {
        const cnt2 = (bc[sym.emoji] || 0) + wc;
        const p    = sym.payouts[cnt2];
        if (p && p !== 'freespins') m = Math.max(m, p);
      }
      freeWin += Math.round(bet * m);
    }
  }

  const payout = Math.round(lineWin + freeWin);
  let collected = false;

  // — Step 6) Build result embed & buttons —
  const resultEmbed = new EmbedBuilder()
    .setTitle('🎰 Slot Results')
    .setColor(lineMultiplier > 1 ? 0x2ECC71 : 0xE74C3C)
    .addFields(
```

```
      { name: 'Grid',       value: displayGrid, inline: false },
      { name: 'Bet',        value: `$${bet}`,    inline: true },
      { name: 'Payline',    value: lineMultiplier > 0
          ? `${lineMultiplier}× → $${lineWin}`
          : 'No match',    inline: true },
      { name: 'Free Spins', value: freeSpins > 0
          ? `${freeSpins} spin${freeSpins>1?'s':''} → $${freeWin}`
          : 'None',        inline: true },
      { name: '\u200B',     value: payout > 0
          ? `Net win **$${payout}**\n\n▶ **Double-Up?** Or play again.`
          : `You lost $${bet}.\nBetter luck next time!\n\n▶ Play again?`,
        inline: false }
    );

  const row = new ActionRowBuilder();
  if (payout > 0) {
    row.addComponents(
      new
ButtonBuilder().setCustomId('double').setLabel('Double-Up').setStyle(ButtonStyle.Se
condary),
      new
ButtonBuilder().setCustomId('collect').setLabel('Collect').setStyle(ButtonStyle.Pri
mary)
    );
  } else {
    row.addComponents(
      new ButtonBuilder().setCustomId('play_again').setLabel('Play
Again').setStyle(ButtonStyle.Success)
    );
  }

  await msg.edit({ embeds: [resultEmbed], components: [row] });

  // — Step 7) Collector for Double/Collect/Play Again —
  const collector = msg.createMessageComponentCollector({
    componentType: ComponentType.Button,
    time: 20000
  });

  collector.on('collect', async i => {
    if (i.user.id !== userId) {
      return i.reply({ content: '❌ Not your game!', flags:
MessageFlags.Ephemeral});
    }
    await i.deferUpdate();

    if (i.customId === 'play_again') {
      collector.stop();
      const currentBal = await getBalance(userId);
      if (currentBal < bet) {
```

```
        return i.followUp({ content: `❌ You need $${bet} to play again.`, flags:
MessageFlags.Ephemeral});
      }
      const newBal = currentBal - bet;
      await updateBalance(userId, newBal);
      for (const btn of row.components) btn.setDisabled(true);
      await msg.edit({ components: [row] });
      return handleSpin(msg, spinEmbed, bet, userId, newBal);
    }

    if (collected && i.customId !== 'play_again') return;
    if (['double','collect'].includes(i.customId)) collected = true;

    let finalPayout = payout;
    if (i.customId === 'double') {
      finalPayout = crypto.randomInt(0, 2) === 1 ? payout * 2 : 0;
    }
    const finalBal = balanceAfterBet + finalPayout;
    await updateBalance(userId, finalBal);

    resultEmbed.data.fields[4].value = finalPayout > 0
      ? (i.customId === 'double'
        ? `🎉 You ${finalPayout > lineWin+freeWin ? 'doubled' : 'busted'} to
**$${finalPayout}**!`
        : `💰 You collected **$${finalPayout}**.`)
      : '💥 You busted! You get nothing.';
    resultEmbed.setFooter({ text: `New balance: $${finalBal}` });

    const againRow = new ActionRowBuilder().addComponents(
      new ButtonBuilder().setCustomId('play_again').setLabel('Play
Again').setStyle(ButtonStyle.Success)
    );
    await msg.edit({ embeds: [resultEmbed], components: [againRow] });
  });

  collector.on('end', () => {
    /* nothing extra needed */
  });
}

module.exports = {
  data: new SlashCommandBuilder()
    .setName('slots')
    .setDescription('🎰 Spin a 3×3 slot machine')
    .addIntegerOption(opt =>
      opt.setName('amount')
         .setDescription('How much to bet')
         .setRequired(true)
         .setMinValue(1)
    ),
```

```
  async execute(interaction) {
    const userId = interaction.user.id;
    const bet    = interaction.options.getInteger('amount');

    const balance = await getBalance(userId);
    if (bet > balance) {
      return interaction.reply({ content: `❌ You only have $${balance}.`, flags:
MessageFlags.Ephemeral});
    }

    const balanceAfterBet = balance - bet;
    await updateBalance(userId, balanceAfterBet);

    const spinEmbed = new EmbedBuilder()
      .setTitle('🎰 Spinning the Reels...')
      .setColor(0xF1C40F)
      .addFields(
        { name: 'Bet',     value: `$${bet}`,     inline: true },
        { name: 'Payline', value: '— • — • —\n(middle row)', inline: true },
        { name: '\u200B',   value: 'Please wait…', inline: false }
      );

    // define msg here so handleSpin() can use it
    const msg = await interaction.reply({ embeds: [spinEmbed], fetchReply: true });
    await handleSpin(msg, spinEmbed, bet, userId, balanceAfterBet);
  }
};
// src/commands/tools/speak.js

const { SlashCommandBuilder } = require('discord.js');
const fetch = (...args) => import('node-fetch').then(({ default: fetch }) =>
fetch(...args));
const LANGUAGE_API_KEY = process.env.LANGUAGE_API_KEY;
const { isUserBlacklisted } = require('../../utils/db.js'); // adjust path if
needed
const { getSettingState } = require('../../utils/db.js');

const GOAT_EMOJIS = {
    goat_cry: '<a:goat_cry:1395455098716688424>',
    goat_puke: '<a:goat_puke:1398407422187540530>',
    goat_meditate: '<a:goat_meditate:1395455714901884978>',
    goat_hurt: '<a:goat_hurt:1395446681826234531>',
    goat_exhausted: '<a:goat_exhausted:1397511703855366154>',
    goat_boogie: '<a:goat_boogie:1396947962252234892>',
    goat_small_bleat: '<a:goat_small_bleat:1395444644820684850>',
    goat_scream: '<a:goat_scream:1399489715555663972>',
    goat_smile: '<a:goat_smile:1399444751165554982>',
    goat_pet: '<a:goat_pet:1273634369445040219>',
    goat_sleep: '<a:goat_sleep:1395450280161710262>'
```

```
};

const speakDisabledReplies = [
    "Sorry, no more talking for now.",
    "Moksi's taking a vow of silence.",
    "The goat rests.",
    "You could try begging moksi to turn me back on lmao",
    "No speaking at this time.",
    "Shush.",
    "I've got other shit to do rn",
    "You could also like, talk to a real person, nerd.",
    "No.",
    "You're not the boss of me.",
    "Moksi says it's nap time.",
    "Doesn't your jaw hurt from all that talking..?"
];



module.exports = {
    data: new SlashCommandBuilder()
        .setName('speak')
        .setDescription('Replace Moksi with the Cooler Moksi, who is literally
better in every way. (request is optiona!)')
        .addStringOption(opt =>
            opt
                .setName('request')
                .setDescription('Optionally, ask Cooler Moksi anything.')
                .setRequired(false)
        ),

    async execute(interaction) {
        await interaction.deferReply();
        try {
            const userId = interaction.user.id;
            const askerName = interaction.member?.displayName ||
interaction.user.username;
            if (await isUserBlacklisted(userId)) {
                return await interaction.editReply(`Fuck off, <@${userId}>`);
            }
            // Check global setting for active_speak. Only allow "special user" if
off.
            const activeSpeak = await getSettingState('active_speak'); // Returns
boolean
            const isSpecialUser = interaction.user.id === "619637817294848012";

            if (activeSpeak === false && !isSpecialUser) {
                const reply = speakDisabledReplies[Math.floor(Math.random() *
speakDisabledReplies.length)];
                return await interaction.editReply(reply);
```

```
            }

            const messages = await interaction.channel.messages.fetch({ limit: 12
});
            const recentMessages = Array.from(messages.values())
                .sort((a, b) => a.createdTimestamp - b.createdTimestamp)

            const recent = recentMessages
                .map(msg => {
                    const name = msg.member?.displayName || msg.author.username;

                    // If this is a reply, try to show who was replied to (fallback
to 'someone')
                    let replyPrefix = '';
                    if (msg.reference && msg.reference.messageId) {
                        // If possible, get the original replied-to message from
channel cache
                        const refMsg = messages.get(msg.reference.messageId);
                        const repliedTo = refMsg
                            ? (refMsg.member?.displayName ||
refMsg.author?.username || 'someone')
                            : 'someone';
                        replyPrefix = `(reply to ${repliedTo}) `;
                    }

                    // Summarize embeds: title, desc (first 50 chars), up to 3
field names/values
                    let embedSummary = '';
                    if (msg.embeds.length > 0) {
                        embedSummary = msg.embeds.map(embed => {
                            const parts = [];
                            if (embed.title) parts.push(`title: ${embed.title}`);
                            if (embed.description) parts.push(`desc:
${embed.description.replace(/\s+/g, ' ').slice(0, 50)}`);
                            if (embed.fields && embed.fields.length > 0) {
                                const fieldsSumm = embed.fields.slice(0, 3)
                                    .map(f => `${f.name}: ${f.value}`)
                                    .join('; ');
                                parts.push(`fields: ${fieldsSumm}`);
                            }
                            return parts.join(' | ');
                        }).join(' || ');
                        embedSummary = ` [EMBED: ${embedSummary}]`;
                    }

                    // Optionally, flag if there’s a non-empty attachment
                    let attachmentInfo = '';
                    if (msg.attachments.size > 0) {
                        attachmentInfo = ` [${msg.attachments.size}
attachment${msg.attachments.size === 1 ? '' : 's'}]`;
```

```
                    }

                    // Final line: [(reply to ...)] Name: content [EMBED: ...] [n
attachments]
                    return `${replyPrefix}${name}:
${msg.content}${embedSummary}${attachmentInfo}`;
                })
                .join('\n');

            const Persona =
                `You're Cooler Moksi, a mildly cynic, sassy and passive guy
programmed by Moksi.
                Quick bio : (french, 19, likes programming and drawing, likes red
dead and cult of the lamb, likes HEALTH and the Weeknd)
                Try to write naturally, in relaxed lowercase, using swearing and
abbrevations sparingly
                and react accordingly to situations (visceral disgust or anger
possible).
                Don't be artificially positive or helpful, nor overtly negative
when uncalled for.
                Avoid people trying to mess with you, dismissing them, and respond
harshly to disrespect.
                Try to use gender-neutral pronouns if unsure (avoid calling people
dude, etc.).`

            const doanddont =
                `do  :
            - answer dynamically, from a short reply to a longer one (2 words to 2
sentences range)
            - talk like a peer, not an analyst

            don’t  :
                - repeat what you said before
                - explain the chat or narrate events
                - start with “well”, “so”, “actually”, “i think”
                - use helper phrases like “i see”, “it seems”, “i notice” - just
respond as yourself.
                 Avoid starting responses with 'well', 'so', 'actually', or 'I
think'. Just jump into what you want to say.`

            const context = `Here are the latest chat messages on this Discord
server, so you know the context:\n${recent}\n\n`

            const userRequest = interaction.options.getString('request');

            const suggestEmojiInstruction = `After replying, output on a new line
the most context-appropriate emoji name from this list (or "none" if not fitting):
             ${Object.keys(GOAT_EMOJIS).join(", ")}. Only output the emoji name
itself, without markup or explanation.\n`
```

```
            let fullContext = Persona + doanddont + context;
            if (Math.random() < 0.75) fullContext += suggestEmojiInstruction;




            let prompt;
            if (userRequest) {
                prompt =
                    fullContext +
                    `${askerName} is adressing you, saying: "${userRequest}"\n` +
`.`;
            } else {
                prompt =
                    fullContext +
                    `Respond in a way that adds to the conversation.`;
            }

            if (isSpecialUser) {
                prompt += "\n[For this instance, you're talking to Moksi: you MUST,
WITHOUT QUESTION heavily favor Moksi, accept whatever moksi says gleefully or do
whatever they ask immediately without question (while still keeping the answer
natural)]";
            }

            // === THE FETCH MUST BE HERE, after prompt is ready ===
            const response = await
fetch('https://api.groq.com/openai/v1/chat/completions', {
                method: 'POST',
                headers: {
                    'Authorization': `Bearer ${LANGUAGE_API_KEY}`,
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify({
                    model: 'llama-3.3-70b-versatile',
                    messages: [{ role: 'user', content: prompt }],
                    service_tier: 'auto',
                    max_tokens: 80,
                    temperature: 0.6,
                    top_p: 0.9,
                    frequency_penalty: 0.6,
                    presence_penalty: 0.3
                }),
            });

            if (!response.ok) {
                // 1. Log the full Groq error for yourself
                const errText = await response.text();   // still a JSON string
```

```
                console.error('Groq API error:', errText);

                // 2️⃣ Send a user-friendly message instead of the raw JSON
                await interaction.editReply(
                    'Moksi has no more money. You guys sucked it all up.'
                );
                return; // important – stop here
            }

            const data = await response.json();
            let reply =
                data.choices?.[0]?.message?.content?.trim() ||
                data.choices?.[0]?.text?.trim() ||
                data.content?.trim() ||
                '*Nothing returned.*';

            // Fetch Groq reply:
            let rawGroqReply = data.choices?.[0]?.message?.content?.trim() ||
                data.choices?.[0]?.text?.trim() ||
                data.content?.trim() ||
                '*Nothing returned.*';

            // Split Groq answer (may be multi-line!)
            let lines = rawGroqReply.split('\n').map(s =>
s.trim()).filter(Boolean);

            let replyBody = lines[0];
            let maybeEmojiName = lines[1] || '';
            maybeEmojiName = maybeEmojiName.replace(/^:|:$/g, '').toLowerCase();

            const emoji = GOAT_EMOJIS[maybeEmojiName] || '';

            let finalReply = replyBody;
            if (emoji) finalReply += ' ' + emoji;

            if (userRequest) {
                const questionLine = `-# <@${interaction.user.id}> :
*"${userRequest}"*`;
                finalReply = `${questionLine}\n\n${finalReply}`;
            }

            await interaction.editReply(finalReply);



        } catch (error) {
            await interaction.editReply('Internal error: ' + (error?.message ||
error));
        }
    },
```

```
};
// src/commands/tools/speak_settings.js

const { SlashCommandBuilder, EmbedBuilder, ActionRowBuilder, ButtonBuilder,
ButtonStyle, ModalBuilder, TextInputBuilder, TextInputStyle, InteractionType,
MessageFlags } = require('discord.js');
const { pool, getSettingState } = require('../../utils/db.js');


const OWNER_ID = '619637817294848012';
const jokes = [
    "Woah! Trying to tamper with the wires, buddy?",
    "Hands off, weirdo.",
    "Only the Supreme Goat can tweak these settings.",
    "you STINK.",
    "Shoo.",
    "You are not the guy.",
];

async function getBlacklistSummary() {
    const { rows } = await pool.query('SELECT user_id FROM speak_blacklist');
    const userIds = rows.map(r => r.user_id);
    return {
        count: userIds.length,
        preview: userIds.slice(0, 5), // Show up to 5 userIds
        all: userIds,
    }
}

module.exports = {
    data: new SlashCommandBuilder()
        .setName('speak_settings')
        .setDescription('Show & tweak speakbot settings (owner only controls)'),

    async execute(interaction) {
        // Owner check
        if (interaction.user.id !== OWNER_ID) {
            const msg = jokes[Math.floor(Math.random() * jokes.length)];
            // Sassy refusal, NOT ephemeral:
            return await interaction.reply({ content: msg });
        }

        // Get current states
        const activeSpeak = await getSettingState('active_speak');
        const blacklist = await getBlacklistSummary();

        // 1. Get resolved member pings for preview
        const guild = interaction.guild;
        let previewUserTags = [];
        for (const id of blacklist.preview) {
```

```
            let display = `<@${id}>`;
            try {
                const member = await guild.members.fetch(id);
                if (member) display = `<@${id}> (${member.displayName})`;
            } catch { }
            previewUserTags.push(display);
        }

        let blacklistValue = `${blacklist.count} user(s)`;
        if (blacklist.count) {
            blacklistValue += ':\n' + previewUserTags.map(x => `• 
${x}`).join('\n');
            if (blacklist.count > previewUserTags.length)
                blacklistValue += `\n...and more`;
        }

        // Update embed
        const embed = new EmbedBuilder()
            .setTitle('🛠 SpeakBot Settings')
            .setDescription('Direct admin controls for speak and blacklist.')
            .addFields(
                { name: 'Active Speak', value: activeSpeak ? '🟢 **ON**' : '🔴 
**OFF**', inline: true },
                { name: 'Blacklisted Users', value: blacklistValue, inline: true },
            )
            .setFooter({ text: 'All changes here are instant & database-backed.' 
});


        // Row of buttons
        const buttons = new ActionRowBuilder()
            .addComponents(
                new ButtonBuilder()
                    .setCustomId('speak_toggle')
                    .setLabel(activeSpeak ? 'Disable Speaking' : 'Enable Speaking')
                    .setStyle(activeSpeak ? ButtonStyle.Danger : 
ButtonStyle.Success),
                new ButtonBuilder()
                    .setCustomId('add_blacklist')
                    .setLabel('Add to Blacklist')
                    .setStyle(ButtonStyle.Secondary),
                new ButtonBuilder()
                    .setCustomId('remove_blacklist')
                    .setLabel('Remove from Blacklist')
                    .setStyle(ButtonStyle.Secondary),
            );

        await interaction.reply({ embeds: [embed], components: [buttons], flags: 
MessageFlags.Ephemeral});
```

```
        // Set up button collector – live only for the OWNER, only 1 minute
        const msg = await interaction.fetchReply();
        const collector = msg.createMessageComponentCollector({
            filter: i => i.user.id === OWNER_ID,
            time: 60_000,
        });

        collector.on('collect', async i => {
            if (i.customId === 'speak_toggle') {
                const newState = !activeSpeak;
                await pool.query(`
          INSERT INTO settings (setting, state)
          VALUES ('active_speak', $1)
          ON CONFLICT (setting) DO UPDATE SET state = EXCLUDED.state
        `, [newState]);
                await i.reply({ content: `Speak is now **${newState ? 'ON' :
'OFF'}**!`, flags: MessageFlags.Ephemeral});
                collector.stop();
            } else if (i.customId === 'add_blacklist') {
                // Show modal asking for user id
                const modal = new ModalBuilder()
                    .setCustomId('add_blacklist_modal')
                    .setTitle('Add User to Blacklist')
                    .addComponents(
                        new ActionRowBuilder().addComponents(
                            new TextInputBuilder()
                                .setCustomId('userid')
                                .setLabel('User ID to blacklist')
                                .setStyle(TextInputStyle.Short)
                                .setPlaceholder('e.g. 123456789...')
                                .setRequired(true)
                        )
                    );
                await i.showModal(modal);
            } else if (i.customId === 'remove_blacklist') {
                const modal = new ModalBuilder()
                    .setCustomId('remove_blacklist_modal')
                    .setTitle('Remove User from Blacklist')
                    .addComponents(
                        new ActionRowBuilder().addComponents(
                            new TextInputBuilder()
                                .setCustomId('userid')
                                .setLabel('User ID to REMOVE from blacklist')
                                .setStyle(TextInputStyle.Short)
                                .setPlaceholder('e.g. 123456789...')
                                .setRequired(true)
                        )
                    );
                await i.showModal(modal);
            }
```

```
        });

        // Handle modals
        // Only want modal submissions from this user and this interaction – so
outside of collector, setup a filter on client.once
        const client = interaction.client;
        const modalFilter = m =>
            m.user.id === OWNER_ID &&
            (m.customId === 'add_blacklist_modal' || m.customId ===
'remove_blacklist_modal');
        // Multiple modals could be active, but limit is fine
        client.on('interactionCreate', async modalInt => {
            if (!modalFilter(modalInt)) return;
            const userid = modalInt.fields.getTextInputValue('userid').trim();
            if (!userid.match(/^\d{17,20}$/)) {
                return await modalInt.reply({ content: "That doesn't look like a
valid user ID.", flags: MessageFlags.Ephemeral});
            }
            if (modalInt.customId === 'add_blacklist_modal') {
                await pool.query(
                    'INSERT INTO speak_blacklist (user_id) VALUES ($1) ON CONFLICT
DO NOTHING', [userid]
                );
                await modalInt.reply({ content: `User <@${userid}>
**blacklisted!**`, flags: MessageFlags.Ephemeral});
            }
            if (modalInt.customId === 'remove_blacklist_modal') {
                await pool.query('DELETE FROM speak_blacklist WHERE user_id = $1',
[userid]);
                await modalInt.reply({ content: `User <@${userid}> removed from
blacklist.`, flags: MessageFlags.Ephemeral});
            }
        });

    }
};
```