```
const TelegramBot = require("node-telegram-bot-api");

// Load English TET questions (Paper 1, 31–60) from JSON
// Fields in JSON: question, options, correctIndex, subjectId, categoryId, topicId, explanation, tip (optional), passage (optional)
const questions = require("./eng_paper1_q31_60.json");

// ---------- PREMIUM / FREE CONFIG ----------

// Temporary premium users list (Telegram user IDs)
// 👉 Replace 123456789 with your real Telegram ID
const premiumUsers = new Set([
  437248254, // you
  // add more IDs here as needed
]);

function isPremiumUser(userId) {
  return premiumUsers.has(userId);
}

// Free plan config
const FREE_DAILY_MINI_TESTS = 1; // 1 test per day
const MINI_TEST_SIZE = 5;       // 5 questions per free test

// ---------- BOT SETUP ----------

const bot = new TelegramBot(process.env.BOT_TOKEN, { polling: true });
```

```
// In-memory session store per chat (active tests)
const sessions = {};

// Store last completed test per chat (for review buttons + analytics)
const lastResults = {};

// In-memory user stats for leaderboard + free limit tracking
// Structure:
// userStats[userId] = {
//   id, name, attempts, bestScore, lastScore,
//   lastFreeDate, freeTestsToday
// }
const userStats = {};

const letters = ["a", "b", "c", "d"];
// Fun reaction emojis for instant feedback
const correctReactions = ["🎯", "👏", "🔥", "💯", "🤩"];
const wrongReactions = ["😳", "😅", "📚", "🤔", "💭"];

// Main menu keyboard
const mainMenu = {
  reply_markup: {
    keyboard: [
      ["📘 Daily Practice Test", "📕 Full Mock Test"],
      ["🏆 Leaderboard", "ℹ️ Help"],
    ],
    resize_keyboard: true,
    one_time_keyboard: false,
```

```
  },
};

// ---------- HELPERS ----------

// Small text progress bar like: [▓▓▓▓▓▓▓▓▓▓] 50%
function makeProgressBar(correct, total, length = 10) {
  if (total === 0) return "[----------]";
  const ratio = correct / Math.max(total, 1);
  const filled = Math.round(ratio * length);
  let bar = "[";
  for (let i = 0; i < length; i++) {
    bar += i < filled ? "▓" : "▒";
  }
  bar += "]";
  return bar;
}

// Build nice display name
function getDisplayName(user) {
  if (user.username) return "@" + user.username;
  const parts = [user.first_name, user.last_name].filter(Boolean);
  if (parts.length) return parts.join(" ");
  return `User_${user.id}`;
}

// Get explanation preview (1st sentence or ~120 chars)
function getExplanationPreview(full) {
```

```
 if (!full || typeof full !== "string") return "";
 const trimmed = full.trim();
 const dotIndex = trimmed.indexOf(".");
 if (dotIndex > 20 && dotIndex < 160) {
  return trimmed.slice(0, dotIndex + 1);
 }
 if (trimmed.length <= 120) return trimmed;
 return trimmed.slice(0, 120) + "...";
}

// ---------- TEST FLOW ----------

// Start a new test with a given pool
function startTest(chatId, user, questionsPoolOverride, isFreeMini = false) {
 const pool = questionsPoolOverride || questions;

 sessions[chatId] = {
  currentIndex: 0,
  score: 0,
  answers: [],
  user: {
   id: user.id,
   username: user.username,
   first_name: user.first_name,
   last_name: user.last_name,
  },
  isWrongRetake: false,
  isFreeMini,
```

```
    questionsPool: pool,
  };

  sendQuestion(chatId);
}

// Start a test only for wrong answers from last test (Mentor+ only)
function startWrongRetake(chatId, user) {
  const prevResult = lastResults[chatId];
  if (!prevResult || !prevResult.answers || prevResult.answers.length === 0) {
    bot.sendMessage(
      chatId,
      "❗ No recent test data found.\nTake a Daily Practice Test first.",
      mainMenu,
    );
    return;
  }

  const basePool = prevResult.questionsPool || questions;
  const wrongAnswers = prevResult.answers.filter((a) => !a.correct);

  const uniqueIndices = Array.from(new Set(wrongAnswers.map((a) => a.qIndex)));

  const wrongPool = uniqueIndices
    .map((idx) => basePool[idx])
    .filter((q) => Boolean(q));

  if (!wrongPool.length) {
```

```
  bot.sendMessage(
    chatId,
    "🎉 You got everything correct in the last test.\nNo wrong-only retest needed!",
    mainMenu,
  );
  return;
}

sessions[chatId] = {
  currentIndex: 0,
  score: 0,
  answers: [],
  user: {
    id: user.id,
    username: user.username,
    first_name: user.first_name,
    last_name: user.last_name,
  },
  isWrongRetake: true,
  isFreeMini: false,
  questionsPool: wrongPool,
};

bot.sendMessage(
  chatId,
  "🎯 Starting a test with only your previous *wrong* questions...",
  { parse_mode: "Markdown" },
);
```

```
 sendQuestion(chatId);
}

// Start daily practice test (respects free vs Mentor+ limits)
function startDailyPracticeTest(chatId, user) {
 const userId = user.id;

 // 💎 Mentor+ users → unlimited tests, full pool
 if (isPremiumUser(userId)) {
  // 🔕 No extra message here – go straight to questions
  startTest(chatId, user, questions, false);
  return;
 }

 // 🆓 Free users → 1 mini-test (5 questions) per day
 const today = new Date().toISOString().slice(0, 10);

 if (!userStats[userId]) {
  userStats[userId] = {
   id: userId,
   name: getDisplayName(user),
   attempts: 0,
   bestScore: 0,
   lastScore: 0,
   lastFreeDate: null,
   freeTestsToday: 0,
  };
 }
```

```
const stats = userStats[userId];

// Reset daily counter if new day
if (stats.lastFreeDate !== today) {
 stats.lastFreeDate = today;
 stats.freeTestsToday = 0;
}

if (stats.freeTestsToday >= FREE_DAILY_MINI_TESTS) {
 bot.sendMessage(
  chatId,
  "🚧 *Free limit reached for today*\n\n" +
   `You already used your free ${MINI_TEST_SIZE}-question practice test.\n\n` +
   "📝 Free plan:\n" +
   `• 1 mini-test (${MINI_TEST_SIZE} questions) per day\n\n` +
   "💎 Use /status or tap *🔐 Unlock full access* after a test to upgrade to Mentor+.",
  { parse_mode: "Markdown", ...mainMenu },
 );
 return;
}

stats.freeTestsToday += 1;

const dailyPool = questions.slice(0, MINI_TEST_SIZE); // can randomise later

bot.sendMessage(
 chatId,
```

```
    `📘 Starting today's *free* ${MINI_TEST_SIZE}-question practice test...`,
    { parse_mode: "Markdown" },
  );

  startTest(chatId, user, dailyPool, true);
}


// Send current question (handles passage separately to avoid auto-scroll to bottom)
function sendQuestion(chatId) {
  const session = sessions[chatId];
  if (!session) return;

  const pool = session.questionsPool || questions;
  const qIndex = session.currentIndex;
  if (qIndex >= pool.length) {
    sendResult(chatId);
    return;
  }

  const q = pool[qIndex];
  const total = pool.length;

  let text = `Q${qIndex + 1}/${total}\n\n`;

  // 1️⃣ Show passage first (if any)
  if (q.passage && typeof q.passage === "string" && q.passage.trim().length > 0) {
    text += `📖 *Passage/Poem:*\n${q.passage}\n\n`;
```

```
}

// 2️⃣ Then question
text += `${q.question}\n\n`;

// 3️⃣ Then options
q.options.forEach((opt, i) => {
 text += `${letters[i]}) ${opt}\n`;
});

text += `\nChoose one option 👇`;

const inlineKeyboard = [
 [
  { text: "a", callback_data: `${qIndex}:0` },
  { text: "b", callback_data: `${qIndex}:1` },
  { text: "c", callback_data: `${qIndex}:2` },
  { text: "d", callback_data: `${qIndex}:3` },
 ],
 [
  { text: "⏭️ Skip", callback_data: `skip:${qIndex}` },
  { text: "🏁 Finish test now", callback_data: `finish_now:${qIndex}` },
 ],
];

// ❗ IMPORTANT: only ONE sendMessage here
bot.sendMessage(chatId, text, {
```

```
    parse_mode: "Markdown",
    reply_markup: { inline_keyboard: inlineKeyboard },
  });
}

// ---------- TOPIC ANALYTICS HELPERS ----------

function calculateTopicStats(result) {
  const topicStats = {}; // key: subject|category|topic

  result.answers.forEach((a) => {
    const subjectId = a.subjectId || "NA_SUBJ";
    const categoryId = a.categoryId || "NA_CAT";
    const topicId = a.topicId || "NA_TOPIC";

    const key = `${subjectId}|${categoryId}|${topicId}`;
    if (!topicStats[key]) {
      topicStats[key] = {
        subjectId,
        categoryId,
        topicId,
        attempted: 0,
        correct: 0,
      };
    }
    topicStats[key].attempted++;
    if (a.correct) topicStats[key].correct++;
  });
```

```
  return topicStats;
}

function getWeakTopics(topicStats, threshold = 60, minAttempt = 2) {
  const weak = [];

  Object.values(topicStats).forEach((stat) => {
    if (stat.attempted < minAttempt) return;
    const accuracy = (stat.correct / stat.attempted) * 100;
    if (accuracy < threshold) {
      weak.push({
        ...stat,
        accuracy: Math.round(accuracy),
      });
    }
  });

  weak.sort((a, b) => a.accuracy - b.accuracy);
  return weak;
}

// ---------- SUMMARY & REVIEW TEXT ----------

function formatSummaryMessage(result) {
  const pool = result.questionsPool || questions;
  const totalQuestions = pool.length;
  const attempted = result.answers.length;
```

```
  const correct = result.answers.filter((a) => a.correct).length;
  const wrong = attempted - correct;
  const skipped = totalQuestions - attempted;
  const accuracy = attempted > 0 ? Math.round((correct / attempted) * 100) : 0;
  const bar = makeProgressBar(correct, attempted, 10);

  let msg = "";
  msg += "📄 *Test finished!*\n\n";
  msg += "📊 *Summary*\n\n";
  msg += `📌 *Score:* ${correct}/${attempted}\n`;
  msg += `📝 *Attempted:* ${attempted}/${totalQuestions}\n`;
  msg += `⏭️ *Skipped:* ${skipped}\n`;
  msg += `❌ *Wrong:* ${wrong}\n`;
  msg += `🎯 *Accuracy:* ${accuracy}% (on attempted)\n\n`;
  msg += `📈 Progress: ${bar}\n`;

  if (accuracy < 40) {
    msg += "\n🔴 Focus more on basics. Revise core concepts slowly.\n";
  } else if (accuracy < 70) {
    msg += "\n🟡 Good attempt. Strengthen the wrong ones with targeted practice.\n";
  } else {
    msg += "\n🟢 Nice work! Just polish the remaining weak spots.\n";
  }

  return msg;
}
```

```
// Full right-answers message (Mentor+)
function formatRightAnswersMessage(result) {
 const pool = result.questionsPool || questions;
 const rightAnswers = result.answers.filter((a) => a.correct);

 if (!rightAnswers.length) {
  return "ℹ️ You had no fully correct answers in this test.";
 }

 let text = "✅ *Right Answers (with explanations)*\n\n";

 rightAnswers.forEach((ans, idx) => {
  const q = pool[ans.qIndex];
  if (!q) return;

  const correctOption = q.options[q.correctIndex];
  const correctLetter = letters[q.correctIndex];

  text += `Q${idx + 1}) ${q.question}\n`;
  text += `✅ *Correct:* ${correctLetter}) ${correctOption}\n`;

  if (q.explanation) {
   text += "🔎 *Why this is correct?*\n";
   text += `• ${q.explanation}\n`;
  }
  text += "\n";
 });
```

```
  text +=
    "➡️ You can now check wrong answers, see topic-wise performance, or retake only wrong questions.";
  return text;
}

// Full wrong-answers message (Mentor+)
function formatWrongAnswersMessage(result) {
  const pool = result.questionsPool || questions;
  const wrongAnswers = result.answers.filter((a) => !a.correct);

  if (!wrongAnswers.length) {
    return "🎉 You had no wrong answers in this test.\nExcellent work!";
  }

  let text = "❌ *Wrong Answers (with explanations)*\n\n";

  wrongAnswers.forEach((ans, idx) => {
    const q = pool[ans.qIndex];
    if (!q) return;

    const correctOption = q.options[q.correctIndex];
    const correctLetter = letters[q.correctIndex];

    const chosenOption =
      ans.chosen != null ? q.options[ans.chosen] : "No option selected";
    const chosenLetter = ans.chosen != null ? letters[ans.chosen] : "-";
```

```
  text += `Q${idx + 1}) ${q.question}\n`;
  text += `❌ *Your answer:* ${chosenLetter}) ${chosenOption}\n`;
  text += `✅ *Correct:* ${correctLetter}) ${correctOption}\n`;

  if (q.explanation) {
   text += "🔎 *Why this is correct?*\n";
   text += `• ${q.explanation}\n`;
  }

  if (q.tip) {
   text += "👳 *Teaching tip:*\n";
   text += `• ${q.tip}\n`;
  }

  text += "\n";
 });

 text +=
  "➡️ You can now retake only these wrong questions, or go back to main menu.";
 return text;
}

// Wrong-answers PREVIEW (Free users: 1st line teaser)
function formatWrongAnswersPreviewMessage(result) {
 const pool = result.questionsPool || questions;
 const wrongAnswers = result.answers.filter((a) => !a.correct);
```

```
if (!wrongAnswers.length) {
  return (
    "🎉 You had no wrong answers in this test.\n\n" +
    "For detailed explanations & tips for every question, unlock Mentor+."
  );
}

let text = "❌ *Wrong Answers (preview)*\n\n";

wrongAnswers.forEach((ans, idx) => {
  const q = pool[ans.qIndex];
  if (!q) return;

  const correctOption = q.options[q.correctIndex];
  const correctLetter = letters[q.correctIndex];

  const chosenOption =
    ans.chosen != null ? q.options[ans.chosen] : "No option selected";
  const chosenLetter = ans.chosen != null ? letters[ans.chosen] : "-";

  text += `Q${idx + 1}) ${q.question}\n`;
  text += `❌ *Your answer:* ${chosenLetter}) ${chosenOption}\n`;
  text += `✅ *Correct:* ${correctLetter}) ${correctOption}\n`;

  if (q.explanation) {
    const preview = getExplanationPreview(q.explanation);
    text += "🔎 *Explanation (preview):*\n";
```

```
    text += `• ${preview}\n`;
    text += "🔒 Full explanation + teaching tips available in *Mentor+*.\n";
   } else {
    text += "🔎 *Explanation:* (not added yet)\n";
   }

   text += "\n";
  });

  text +=
   "💡 Use /status to see what Mentor+ unlocks.\n" +
   "Tap *🔐 Unlock full access* below to get complete explanations.";
  return text;
}

function formatTopicStatsMessage(result) {
  const topicStats = result.topicStats || calculateTopicStats(result);
  const entries = Object.values(topicStats);

  if (!entries.length) {
   return "📂 *Topic-wise performance*\n\nNot enough data to show topic-wise stats.";
  }

  let text = "📂 *Topic-wise performance*\n\n";

  entries.forEach((stat) => {
   const { subjectId, categoryId, topicId, attempted, correct } = stat;
```

```
  const accuracy = Math.round((correct / attempted) * 100);

  text += `• *${subjectId}* → _${categoryId}_ → ${topicId}\n`;
  text += `  ${correct}/${attempted} correct (${accuracy}%)\n\n`;
 });

 text +=
  "💡 Use this to see which areas you are consistently strong/weak in across questions.";
 return text;
}

function formatWeakTopicsMessage(result) {
 const topicStats = result.topicStats || calculateTopicStats(result);
 const weakTopics = result.weakTopics || getWeakTopics(topicStats, 60, 2);

 if (!weakTopics.length) {
  return (
   "💪 *Weak topics*\n\n" +
   "🎉 No clear weak topics detected (based on attempts & accuracy threshold). Still, keep revising to maintain your level."
  );
 }

 let text = "💪 *Weak topics (focus here first)*\n\n";

 weakTopics.forEach((w) => {
  text += `• *${w.subjectId}* → _${w.categoryId}_ → ${w.topicId}\n`;
```

```
  text += `  ${w.correct}/${w.attempted} correct (${w.accuracy}%)\n\n`;
 });

 text +=
  "📌 Tip: Focus revision on these topics first, then move to strong areas.";
 return text;
}

// Keyboard after test summary
function buildReviewKeyboard(isPremium, hasWrong) {
 if (isPremium) {
  const inlineKeyboard = [
   [
    { text: "✅ Right answers", callback_data: "view_right" },
    { text: "❌ Wrong answers", callback_data: "view_wrong" },
   ],
   [
    { text: "📂 Topic-wise performance", callback_data: "view_topics" },
    { text: "💪 Weak topics", callback_data: "view_weak_topics" },
   ],
  ];
  if (hasWrong) {
   inlineKeyboard.push([
    { text: "🎯 Retake wrong questions only", callback_data: "retake_wrong" },
   ]);
  }
  inlineKeyboard.push([
```

```
    { text: "🏠 Main Menu", callback_data: "done_results" },
   ]);
   return { inline_keyboard: inlineKeyboard };
  }

  // Free users: preview + upgrade
  const inlineKeyboard = [
   [
    { text: "❌ Wrong answers (preview)", callback_data: "view_wrong" },
    { text: "🔐 Unlock full access", callback_data: "upgrade_mentor" },
   ],
   [
    { text: "🏠 Main Menu", callback_data: "done_results" },
   ],
  ];
  return { inline_keyboard: inlineKeyboard };
}

// ---------- RESULT & LEADERBOARD ----------

function sendResult(chatId) {
  const session = sessions[chatId];
  if (!session) return;

  const pool = session.questionsPool || questions;
  const total = pool.length;
  const score = session.score;
```

```
const user = session.user;
const userId = user.id;
const name = getDisplayName(user);
const isPrem = isPremiumUser(userId);

// Update leaderboard only for normal tests, not wrong-only retake
if (!session.isWrongRetake) {
 if (!userStats[userId]) {
  userStats[userId] = {
   id: userId,
   name,
   attempts: 0,
   bestScore: 0,
   lastScore: 0,
   lastFreeDate: null,
   freeTestsToday: 0,
  };
 }

 const stats = userStats[userId];
 stats.name = name;
 stats.attempts += 1;
 stats.lastScore = score;
 if (score > stats.bestScore) {
  stats.bestScore = score;
 }
}
```

```
const baseResult = {
 answers: session.answers,
 questionsPool: pool,
};
const topicStats = calculateTopicStats(baseResult);
const weakTopics = getWeakTopics(topicStats, 60, 2);

lastResults[chatId] = {
 ...baseResult,
 topicStats,
 weakTopics,
};

const summaryText = formatSummaryMessage(lastResults[chatId]);
const hasWrong = lastResults[chatId].answers.some((a) => !a.correct);
const reviewKeyboard = buildReviewKeyboard(isPrem, hasWrong);

bot
 .sendMessage(chatId, summaryText, {
  parse_mode: "Markdown",
  reply_markup: reviewKeyboard,
 })
 .catch((err) => {
  console.error("Error sending result summary:", err);
  bot.sendMessage(
   chatId,
   `📄 Test finished!\nScore: ${score}/${total}`,
   mainMenu,
```

```
    );
  });

  delete sessions[chatId];
}

function sendLeaderboard(chatId) {
  const list = Object.values(userStats);
  if (!list.length) {
    bot.sendMessage(
      chatId,
      "🏆 No tests attempted yet.\nBe the first! Tap 📘 Daily Practice Test to start.",
      mainMenu,
    );
    return;
  }

  const sorted = [...list].sort((a, b) => {
    if (b.bestScore !== a.bestScore) return b.bestScore - a.bestScore;
    return b.attempts - a.attempts;
  });

  const top = sorted.slice(0, 10);
  let text = "🏆 Leaderboard – Top performers\n\n";

  top.forEach((u, i) => {
    const badge = isPremiumUser(u.id) ? " 💎 " : "";
```

```
  text += `${i + 1}. ${badge}${u.name} — Best: ${u.bestScore || 0}/${questions.length}, Attempts: ${u.attempts}\n`;
 });

 bot.sendMessage(chatId, text, mainMenu);
}

// ---------- COMMANDS ----------

bot.onText(/\/start/, (msg) => {
 const chatId = msg.chat.id;

 const welcome =
  "🎓 Welcome to *KARTET Mentor* (English TET – Prototype)\n\n" +
  "I offer:\n" +
  "• Daily English PYQ mini-test (5Q free)\n" +
  "• Instant ✅/❌ feedback with emojis\n" +
  "• Clean summary with progress bar\n\n" +
  "💎 *Mentor+ (premium) unlocks:*\n" +
  "• Full tests & mocks\n" +
  "• Detailed explanations & teaching tips\n" +
  "• Topic-wise & weak-topic analysis\n" +
  "• Retake wrong-only tests\n" +
  "• 💎 Badge on leaderboard\n\n" +
  "Use the menu below or commands:\n" +
  "/dailytest – Start daily practice\n" +
  "/leaderboard – See top performers\n" +
  "/status – Check your plan (Free / Mentor+)\n" +
```

```
  "/help – Help & info\n";

 bot.sendMessage(chatId, welcome, { parse_mode: "Markdown", ...mainMenu });
});

bot.onText(/\/help/, (msg) => {
 const chatId = msg.chat.id;
 const help =
  "ℹ️ *Help – KARTET Mentor*\n\n" +
  "Commands:\n" +
  "/start – Show main menu\n" +
  "/dailytest – Start a daily practice test (5Q free)\n" +
  "/leaderboard – View top performers\n" +
  "/status – Check whether you are Free or Mentor+\n\n" +
  "🆓 *Free plan:*\n" +
  `• 1 mini-test (${MINI_TEST_SIZE} questions) per day\n` +
  "• Score + accuracy summary\n" +
  "• Wrong-answers preview (1st line of explanation)\n\n" +
  "💎 *Mentor+ (premium):*\n" +
  "• Unlimited tests & full mocks\n" +
  "• Detailed explanations & teaching tips\n" +
  "• Topic-wise & weak-topic analysis\n" +
  "• Retake wrong-only\n" +
  "• 💎 Badge on leaderboard\n";
 bot.sendMessage(chatId, help, { parse_mode: "Markdown", ...mainMenu });
});

bot.onText(/\/status/, (msg) => {
```

```
 const chatId = msg.chat.id;
 const userId = msg.from.id;
 const isPrem = isPremiumUser(userId);
 const name = getDisplayName(msg.from);
 const status = isPrem ? "💎 Mentor+ (Premium Access)" : "🆓 Free User";

 let message = "👤 *Your Account Status*\n\n";
 message += `👥 Name: *${name}*\n`;
 message += `🔐 Plan: *${status}*\n\n`;

 if (isPrem) {
  message += "✨ You have access to:\n";
  message += "• Unlimited tests & full mocks\n";
  message += "• Detailed explanations & tips\n";
  message += "• Topic-wise & weak-topic analysis\n";
  message += "• Wrong-only practice\n";
  message += "• 💎 Badge on leaderboard\n";
 } else {
  message += "🆓 *Free plan:*\n";
  message += `• 1 mini-test (${MINI_TEST_SIZE} questions) per day\n`;
  message += "• Score + accuracy summary\n";
  message += "• Wrong-answers explanation preview\n\n";
  message += "💡 Upgrade to *Mentor+* to unlock full explanations, topic-wise stats and wrong-only practice.\n";
 }

 bot.sendMessage(chatId, message, { parse_mode: "Markdown" });
});
```

```
bot.onText(/\/dailytest/, (msg) => {
  const chatId = msg.chat.id;
  startDailyPracticeTest(chatId, msg.from);
});

bot.onText(/\/leaderboard/, (msg) => {
  const chatId = msg.chat.id;
  sendLeaderboard(chatId);
});

// ---------- HANDLE MENU BUTTON TEXTS ----------

bot.on("message", (msg) => {
  const chatId = msg.chat.id;
  const text = (msg.text || "").trim();

  if (text.startsWith("/")) return;

  if (text === " 📘 Daily Practice Test") {
    startDailyPracticeTest(chatId, msg.from);
  } else if (text === " 📕 Full Mock Test") {
    bot.sendMessage(
      chatId,
      " 📕 Full mock tests will be available for Mentor+ users soon.\nRight now, use the Daily Practice Test.",
      mainMenu,
    );
```

```
 } else if (text === "🏆 Leaderboard") {
  sendLeaderboard(chatId);
 } else if (text === "ℹ️ Help") {
  const help =
   "ℹ️ *Help – KARTET Mentor*\n\n" +
   "Use the menu:\n" +
   "📘 Daily Practice Test – Start English TET PYQ mini-test\n" +
   "📕 Full Mock Test – Coming soon for Mentor+\n" +
   "🏆 Leaderboard – See top performers\n" +
   "ℹ️ Help – Show this message\n";
  bot.sendMessage(chatId, help, { parse_mode: "Markdown", ...mainMenu });
 }
});

// ---------- CALLBACKS (ANSWERS, SKIP, REVIEW, UPGRADE) ----------

bot.on("callback_query", async (callbackQuery) => {
 try {
  const data = callbackQuery.data;
  const msg = callbackQuery.message;
  const chatId = msg.chat.id;
  const userId = callbackQuery.from.id;
  const isPrem = isPremiumUser(userId);

  // --- Skip current question ---
  if (data.startsWith("skip:")) {
   const session = sessions[chatId];
```

```
if (!session) {
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: "No active test to skip.",
    show_alert: false,
  });
  return;
}

const pool = session.questionsPool || questions;
const [, qIndexStr] = data.split(":");
const pressedIndex = parseInt(qIndexStr, 10);

if (pressedIndex !== session.currentIndex) {
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: "This question is already handled.",
    show_alert: false,
  });
  return;
}

await bot.answerCallbackQuery(callbackQuery.id, {
  text: "⏭️ Skipped. Moving to next question...",
  show_alert: false,
});

session.currentIndex++;
if (session.currentIndex < pool.length) {
  sendQuestion(chatId);
```

```
  } else {
    sendResult(chatId);
  }
  return;
}

// --- Finish test early ---
if (data.startsWith("finish_now:")) {
  const session = sessions[chatId];
  if (!session) {
    await bot.answerCallbackQuery(callbackQuery.id, {
      text: "No active test to finish.",
      show_alert: false,
    });
    return;
  }

  const [, qIndexStr] = data.split(":");
  const pressedIndex = parseInt(qIndexStr, 10);

  if (pressedIndex !== session.currentIndex) {
    await bot.answerCallbackQuery(callbackQuery.id, {
      text: "This question is already handled.",
      show_alert: false,
    });
    return;
  }
```

```
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: "🏁 Finishing test with attempted questions...",
    show_alert: false,
  });

  sendResult(chatId);
  return;
}

// --- Upgrade button ---
if (data === "upgrade_mentor") {
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: "Upgrade info sent.",
    show_alert: false,
  });

  const upg =
    "💎 *Upgrade to Mentor+ (Pilot)*\n\n" +
    "Mentor+ unlocks:\n" +
    "• Unlimited daily tests\n" +
    "• Full mocks (coming soon)\n" +
    "• Detailed explanations & teaching tips\n" +
    "• Topic-wise & weak-topic analysis\n" +
    "• Retake wrong-only tests\n" +
    "• 💎 Badge on leaderboard\n\n" +
    "Right now, upgrade is handled *manually* by the admin.\n\n" +
    "If you're interested:\n" +
    "1️⃣ Message the admin: *\"I want Mentor+ for KARTET\"*\n" +
```

```
    "2️⃣ They will share simple UPI/payment details.\n" +
    "3️⃣ Once confirmed, your Telegram ID will be added to the Mentor+ list.\n\n" +
    "(For testing, just add your Telegram ID into the premiumUsers Set in the code.)";
  await bot.sendMessage(chatId, upg, { parse_mode: "Markdown" });
  return;
}

// --- Review: right answers (full for Mentor+) ---
if (data === "view_right") {
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: isPrem ? "Showing right answers..." : "Mentor+ only feature.",
    show_alert: false,
  });

  const result = lastResults[chatId];
  if (!result) {
    await bot.sendMessage(
      chatId,
      "No recent test found. Take a test first.",
      mainMenu,
    );
    return;
  }

  if (!isPrem) {
    await bot.sendMessage(
      chatId,
```

```
      " 🔒 *Right-answer explanations* are for *Mentor+* users.\nUse /status or tap * 🔐 Unlock full access* below your summary to upgrade.",
       { parse_mode: "Markdown" },
      );
      return;
    }

    const text = formatRightAnswersMessage(result);
    const hasWrong = result.answers.some((a) => !a.correct);
    const keyboard = buildReviewKeyboard(true, hasWrong);
    await bot.sendMessage(chatId, text, {
      parse_mode: "Markdown",
      reply_markup: keyboard,
    });
    return;
  }

  // --- Review: wrong answers (preview for Free, full for Mentor+) ---
  if (data === "view_wrong") {
    await bot.answerCallbackQuery(callbackQuery.id, {
      text: isPrem ? "Showing wrong answers..." : "Showing preview...",
      show_alert: false,
    });

    const result = lastResults[chatId];
    if (!result) {
      await bot.sendMessage(
        chatId,
```

```
    "No recent test found. Take a test first.",
    mainMenu,
   );
   return;
  }

  let text, keyboard;
  if (isPrem) {
   text = formatWrongAnswersMessage(result);
   const hasWrong = result.answers.some((a) => !a.correct);
   keyboard = buildReviewKeyboard(true, hasWrong);
  } else {
   text = formatWrongAnswersPreviewMessage(result);
   const hasWrong = result.answers.some((a) => !a.correct);
   keyboard = buildReviewKeyboard(false, hasWrong);
  }

  await bot.sendMessage(chatId, text, {
   parse_mode: "Markdown",
   reply_markup: keyboard,
  });
  return;
 }

 // --- Topic-wise performance (Mentor+ only) ---
 if (data === "view_topics") {
  await bot.answerCallbackQuery(callbackQuery.id, {
   text: isPrem ? "Showing topic-wise performance..." : "Mentor+ only feature.",
```

```
    show_alert: false,
  });

  const result = lastResults[chatId];
  if (!result) {
    await bot.sendMessage(
      chatId,
      "No recent test found. Take a test first.",
      mainMenu,
    );
    return;
  }

  if (!isPrem) {
    await bot.sendMessage(
      chatId,
      "🔒 *Topic-wise performance* is for *Mentor+* users.\nUse /status or tap * 🔓 Unlock full access* to upgrade.",
      { parse_mode: "Markdown" },
    );
    return;
  }

  const text = formatTopicStatsMessage(result);
  const hasWrong = result.answers.some((a) => !a.correct);
  const keyboard = buildReviewKeyboard(true, hasWrong);
  await bot.sendMessage(chatId, text, {
    parse_mode: "Markdown",
```

```
      reply_markup: keyboard,
    });
    return;
  }

  // --- Weak topics (Mentor+ only) ---
  if (data === "view_weak_topics") {
    await bot.answerCallbackQuery(callbackQuery.id, {
      text: isPrem ? "Showing weak topics..." : "Mentor+ only feature.",
      show_alert: false,
    });

    const result = lastResults[chatId];
    if (!result) {
      await bot.sendMessage(
        chatId,
        "No recent test found. Take a test first.",
        mainMenu,
      );
      return;
    }

    if (!isPrem) {
      await bot.sendMessage(
        chatId,
        "🔒 *Weak-topic analysis* is for *Mentor+* users.\nUse /status or tap * 🔓 Unlock full access* to upgrade.",
        { parse_mode: "Markdown" },
```

```
      );
      return;
    }

    const text = formatWeakTopicsMessage(result);
    const hasWrong = result.answers.some((a) => !a.correct);
    const keyboard = buildReviewKeyboard(true, hasWrong);
    await bot.sendMessage(chatId, text, {
      parse_mode: "Markdown",
      reply_markup: keyboard,
    });
    return;
  }

  // --- Retake wrong-only (Mentor+ only) ---
  if (data === "retake_wrong") {
    await bot.answerCallbackQuery(callbackQuery.id, {
      text: isPrem ? "Starting wrong-only test..." : "Mentor+ only feature.",
      show_alert: false,
    });

    if (!isPrem) {
      await bot.sendMessage(
        chatId,
        "🔒 *Wrong-only retake* is for *Mentor+* users.\nUse /status or tap *🔐 Unlock full access* to upgrade.",
        { parse_mode: "Markdown" },
      );
```

```
    return;
  }

  startWrongRetake(chatId, callbackQuery.from);
  return;
}

// --- Done / back to main menu ---
if (data === "done_results") {
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: "Returning to main menu.",
    show_alert: false,
  });
  await bot.sendMessage(chatId, "🏠 Back to main menu.", mainMenu);
  return;
}

// --- Answer handling (data like "0:2") ---
if (!data.includes(":")) {
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: "Unknown action.",
    show_alert: false,
  });
  return;
}

const session = sessions[chatId];
if (!session) {
```

```
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: "No active test. Type /dailytest to start.",
    show_alert: true,
  });
  return;
}

const [qIndexStr, optIndexStr] = data.split(":");
const qIndex = parseInt(qIndexStr, 10);
const chosenIndex = parseInt(optIndexStr, 10);

if (qIndex !== session.currentIndex) {
  await bot.answerCallbackQuery(callbackQuery.id, {
    text: "This question is already answered.",
    show_alert: false,
  });
  return;
}

const pool = session.questionsPool || questions;
const q = pool[qIndex];
const isCorrect = chosenIndex === q.correctIndex;
if (isCorrect) session.score++;

session.answers.push({
  qIndex,
  chosen: chosenIndex,
  correctIndex: q.correctIndex,
```

```
    correct: isCorrect,
    subjectId: q.subjectId,
    categoryId: q.categoryId,
    topicId: q.topicId,
  });

  const reaction = isCorrect
    ? correctReactions[Math.floor(Math.random() * correctReactions.length)]
    : wrongReactions[Math.floor(Math.random() * wrongReactions.length)];

  await bot.answerCallbackQuery(callbackQuery.id, {
    text: isCorrect ? `✅ Correct! ${reaction}` : `❌ Wrong... ${reaction}`,
    show_alert: false,
  });

  setTimeout(() => {
    const activeSession = sessions[chatId];
    if (!activeSession) return;
    const activePool = activeSession.questionsPool || questions;

    activeSession.currentIndex++;
    if (activeSession.currentIndex < activePool.length) {
      sendQuestion(chatId);
    } else {
      sendResult(chatId);
    }
  }, 700);
} catch (err) {
```

```
    console.error("Callback error:", err);
  }
});

console.log(
  "✅ KARTET Mentor bot (English TET, freemium + Mentor+, previews, topic analytics) is running.",
);p
```