**/ * ========== * ЕВА - ИНТЕЛЛЕКТУАЛЬНЫЙ ГОЛОСОВОЙ ПОМОЩНИК * Файл: logic.js * Назначение: Главный диспетчер ядра. Управление состояниями, маршрутизация * речевых конвейеров, автосохранение, таймеры и интеграция баз JSON * Версия: v5.8.2 (Ultra-Safe Tagging & Data Folder Edition) * ========== */ ========== БЛОК 1: ИНИЦИАЛИЗАЦИЯ И СИНХРОНИЗАЦИЯ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ СЕССИИ ========== Флаг ожидания выбора пользователя: true, если Ева ждет команды (Поищи / Ответ / Отмена) window.isWaitingDecision = window.isWaitingDecision || false; *Временное хранилище вопроса пользователя, на который отсутствует ответ в локальной базе window.pendingQuestion = window.pendingQuestion || «»;* Хранилище вопроса для механизма прямого интерактивного пошагового обучения window.lastQ = window.lastQ || «»; *Флаг блокировки: true, пока Ева воспроизводит длинный текст (статью), полученный из интернета window.isSpeakingWiki = window.isSpeakingWiki || false;* Флаг активности речевого синтезатора: true, когда Ева физически произносит фразу через TTS window.isEvaSpeaking = window.isEvaSpeaking || false; *Глобальный объект для хранения структурированных массивов данных из баз JSON window.allData = window.allData || { base: [], aiBase: [], wiki: [], settings: {} };* Глобальное время ожидания проактивности Евы по умолчанию (в миллисекундах) window.IDLE_TIME = window.IDLE_TIME || 30000; ========== БЛОК 2: ВСПОМОГАТЕЛЬНЫЕ И СЛУЖЕБНЫЕ МЕТОДЫ (ПОЛИФИЛЫ РЕЧЕВОГО ТРАКТА) ========== / * *Проверка и регистрация глобальной функции синтеза речи. * Интегрировано с локальным движком Piper TTS через бэкенд на Synology NAS. * @param {string} text - Строка текста, которую необходимо озвучить. */ if (typeof window.speak !== 'function') { window.speak = function(text) { console.log(«[TTS Попытка вызова полифила Piper]:», text); if (!text) return; Прерываем любой текущий звук Piper, чтобы фразы не накладывались друг на друга if (window.evaAudioPlayer) { window.evaAudioPlayer.pause(); } Включаем глобальный флаг активности речи window.isEvaSpeaking = true; Формируем запрос к вашему рабочему бэкенду tts.php на Synology const ttsApiUrl = `/ai/tts.php?text=${encodeURIComponent(text)}`; window.evaAudioPlayer = new Audio(ttsApiUrl); Функция сброса флага при окончании звука или ошибке const resetSpeechState = () ⇒ { window.isEvaSpeaking = false; }; Назначаем триггеры изменения глобального состояния активности речи window.evaAudioPlayer.onended = resetSpeechState; window.evaAudioPlayer.onerror = resetSpeechState; Передаем объект аудио в плеер браузера window.evaAudioPlayer.play().catch*[1)]

1)

err) ⇒ {

```
            console.warn("[TTS  Ошибка]: Воспроизведение заблокировано политикой браузера. Нужен клик.");
```

```
            resetSpeechState();
        });
    };
```

}
==========
========== БЛОК 3: УПРАВЛЕНИЕ ЖИЗНЕННЫМ ЦИКЛОМ ПРИЛОЖЕНИЯ (МЕНЕДЖЕР ЗАПУСКА STARTAPP)
==========
========== ***/ * Главная асинхронная функция инициализации Евы. * Выполняет каскадную загрузку локальных баз знаний из папки data/и подготавливает UI. */ async function startApp() { try { console.log(«[Инициализация]: Запуск загрузки баз данных из папки data/…»); Асинхронно запрашиваем чтение всех файлов через внешний модуль Database из папки data/ const allData = await Database.loadAll(); Проверяем валидность структуры полученного объекта и наличие основной базы if (allData && allData.base) { КРИТИЧЕСКИЙ ШАГ: Дублируем данные в глобальное окно для доступности из любой точки системы window.allData = allData; Передаем загруженные массивы в аналитический модуль нейросети (ИИ-мозг) AI.setData(allData); ПРИНУДИТЕЛЬНОЕ ОБНОВЛЕНИЕ ЛИЧНОСТИ ИЗ SETTINGS.JSON if (allData.settings) { if (allData.settings.voice && typeof EvaConfig !== 'undefined') { Заменяем сухие захардкоженные параметры голоса в config.js на серверные ползунки EvaConfig.voice.pitch = allData.settings.voice.pitch; EvaConfig.voice.rate = allData.settings.voice.rate; EvaConfig.voice.volume = allData.settings.voice.volume; } СИНХРОНИЗАЦИЯ ВСЕХ 8 КИБЕР-ЭМОЦИЙ И ШЕПОТА ИЗ АДМИНКИ НА ЛЕТУ if (allData.settings.emotions && typeof EvaConfig !== 'undefined') { for (let emoKey in allData.settings.emotions) { if (EvaConfig.emotions[emoKey]) { EvaConfig.emotions[emoKey].pitch = allData.settings.emotions[emoKey].pitch; EvaConfig.emotions[emoKey].rate = allData.settings.emotions[emoKey].rate; EvaConfig.emotions[emoKey].volume = allData.settings.emotions[emoKey].volume; EvaConfig.emotions[emoKey].prefix = allData.settings.emotions[emoKey].prefix; } else { Если эмоции не было в базовом коде (например, страх или шёпот), создаем её на лету EvaConfig.emotions[emoKey] = allData.settings.emotions[emoKey]; } } } } Обновляем счетчик записей на графическом интерфейсе пользователя Visual.updateLog(`READY (${allData.base.length})`); Переключаем визуальное состояние аватара в режим дыхания/ожидания Visual.setState('wait'); Если кнопка инициализации найдена в DOM, активируем её неоновую подсветку if (window.mainBtn) window.mainBtn.className = 'btn-start-neon'; console.log(«[Инициализация]: Успешно завершена. Ева готова к работе.»); } } catch (e) { Логируем критическую ошибку (например, повреждение JSON файла или отсутствие файлов) console.error(«[Критическая ошибка инициализации]:», e); Выводим тревожный статус на главный экран приложения Visual.updateLog(«ERROR»); } }***
==========
========== ***БЛОК 4: ИНТЕГРАЦИЯ И ПАРСИНГ РЕЗУЛЬТАТОВ ИЗ СЕТИ ИНТЕРНЕТ (WIKIPEDIA)***
==========
========== *** / * Обработчик успешного получения данных из Wikipedia. * Выполняет вывод текста на экран, чанк-анимацию бегущей строки и запускает озвучку. * @param {string} responseText - Текст статьи, возвращенный парсером. * @param {string|null} incomingQuestion - Вопрос, на который был осуществлен поиск. */ function onWikiResultReceived(responseText, incomingQuestion = null) {*** Активируем системные блокировки, защищающие от прерывания речи новыми звуками микрофона

```
  window.isSpeakingWiki = true;
  window.isWaitingDecision = true;
```

```
 // Вычисляем итоговый вопрос: берем переданный или вытягиваем из системного буфера ожидания
  const finalQuestion = incomingQuestion || window.pendingQuestion;
  console.log("[Wiki  Модуль]: Запуск обработки статьи. Вопрос:", finalQuestion);
 // 🔥 ИНТЕГРАЦИЯ СЕТЕВЫХ НАСТРОЕК: Извлекаем оранжевые подсказки из файла data/settings.json
  const currentDecisionHints = window.allData?.settings?.hints?.decision_hints || "ПОИЩИ / ПОДУМАЙ / ОТВЕТ... / ОТМЕНА";
 // Принудительно выводим актуальные подсказки на верхний статус-бар экрана Евы
  if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') {
      Visual.updateHints(true, currentDecisionHints);
  }
```

```
 // Добавляем полноценный блок ответа Евы в графическое окно диалога чата
  if (typeof Visual !== 'undefined' && typeof Visual.addMsg === 'function') {
      Visual.addMsg(responseText, 'eva-msg');
  }
 // РЕАЛИЗАЦИЯ БЕГУЩЕЙ СТРОКИ: Дробим текст статьи на массив отдельных слов по пробелам
  const words = responseText.split(" ");
  let currentWordIndex = 0;
```

```
 // Создаем цикличный интервал для симуляции синхронного чтения текста Евой на экране
  const textTickerInterval = setInterval(() => {
    // Условие остановки: массив слов исчерпан или внешняя логика принудительно сбросила флаг блокировки
       // 🔥 АППАРАТНАЯ ЗАЩИТА: Останавливаем интервал только если слова закончились И синтезатор полностью замолчал
      if (currentWordIndex >= words.length || (!window.isSpeakingWiki && !window.speechSynthesis.speaking)) {
          clearInterval(textTickerInterval);
     // Сбрасываем текст живой плашки в исходное состояние тишины
          if (typeof Visual !== 'undefined' && typeof Visual.setLiveText === 'function') {
              Visual.setLiveText("");
          }
          return;
      }
```

```
   // Вырезаем порцию из 4-х слов для плавного отображения на экране
      const chunk = words.slice(currentWordIndex, currentWordIndex + 4);
      const tickerText = chunk.join(" ");
```

```
   // Отправляем сформированную порцию слов в текстовое поле живого вывода (синяя
```

```
плашка статус-бара)
      if (typeof Visual !== 'undefined' && typeof Visual.setLiveText === 'function') {
          Visual.setLiveText(tickerText);
      }
```

```
    // Сдвигаем курсор индекса вперед на размер обработанной порции
      currentWordIndex += 4;
 }, 1500); // Оптимальный интервал чтения порции в полторы секунды
```

```
 // Передаем полную статью на озвучивание в TTS-движок голосовой системы
  speak(responseText);
```

```
 // Валидация данных для запуска фонового автоматического сохранения в энциклопедическую базу wiki_base.json
  if (finalQuestion && finalQuestion.length > 0 && responseText.length > 10) {
      console.log("[Wiki Модуль]: Условия выполнены. Отправка статьи на автосохранение...");
    // Вызываем функцию фонового экспорта данных в базу знаний
      if (typeof autoSaveToWiki === 'function') {
          autoSaveToWiki(finalQuestion, responseText);
      } else {
          console.warn("[Wiki Модуль Предупреждение]: Функция autoSaveToWiki не обнаружена в каскаде скриптов.");
      }
  } else {
    // Если статья пустая или некорректная, превентивно снимаем блокировки и усыпляем микрофон
      window.isSpeakingWiki = false;
      window.isWaitingDecision = false;
      if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') {
          Visual.updateHints(false);
      }
  }
```

```
}
================================================================================ БЛОК 5: ИНТЕРФЕЙСНЫЙ ОБРАБОТЧИК ГЛАВНОЙ УПРАВЛЯЮЩЕЙ КНОПКИ
================================================================================ if (window.mainBtn) { /* Обработка нажатия на центральную неоновую кнопку управления Евой. * Реализует два режима: Вход (Активация микрофона) и Выход (Сортировка и сохранение баз). */ window.mainBtn.onclick = async () => { РЕЖИМ А: Приложение спало (window.isLive === false). Запускаем сессию. if (!window.isLive) { window.isLive = true; Переводим систему в статус «В эфире» window.freeTalk = false; По умолчанию включаем режим обращения по имени Меняем текстовое содержание и стиль кнопки на стоп-режим (красный неон) window.mainBtn.innerText = «ВЫЙТИ И СОХРАНИТЬ»; window.mainBtn.className = 'btn-stop-neon'; Инициализируем аудио-контекст захвата голоса и запускаем анализ частот
```

микрофона if (typeof initVoiceSystem === 'function') initVoiceSystem(); if (typeof startMicLevel === 'function') await startMicLevel(); Извлекаем случайное приветствие Евы из конфигурационного файла проекта const hello = EvaConfig.getRandomGreeting(); speak(hello); Visual.addMsg(hello, 'eva-msg'); Запускаем таймер слежения за активностью пользователя resetIdleTimer(); } else { РЕЖИМ Б: Приложение активно. Пользователь завершает сессию. Сортируем память. window.isLive = false; window.mainBtn.innerText = «СОРТИРОВКА…»; МГНОВЕННЫЙ ЗАПУСК АВТОМАТИЧЕСКОГО СЕРВЕРНОГО БЭКАПА 5 БАЗ ДАННЫХ try { console.log(«[Бэкап Модуль]: Инициирую создание точки восстановления перед выходом…»); const backupResponse = await fetch('admin.php?api=create_backup'); if (backupResponse.ok) { const backupData = await backupResponse.json(); if (backupData.status === 'ok') { console.log(«%c[Бэкап Модуль]: Базы успешно заархивированы. Метка: » + backupData.timestamp, «color: green; font-weight: bold;»); } } } catch (backupErr) { console.warn(«[Бэкап Модуль Предупреждение]: Не удалось создать резервную копию:», backupErr); } Сбрасываем фоновые таймеры и деактивируем медиа-потоки записи звука if (window.idleTimer) clearTimeout(window.idleTimer); if (window.rec) try { window.rec.stop(); } catch(e) {} ИНТЕЛЛЕКТУАЛЬНАЯ СОРТИРОВКА: Разделяем временный буфер памяти AI.memory const wikiData = AI.memory.filter(item ⇒ item.isWiki === true); const baseData = AI.memory.filter(item ⇒ item.isWiki !== true); Очищаем временные маркеры 'isWiki', чтобы они не засоряли итоговые файлы базы данных wikiData.forEach(item ⇒ delete item.isWiki); try { console.log(«[Сохранение]: Запуск параллельной отправки данных на сервер в папку data/…»); Асинхронно отправляем массив базовых знаний на обработку скрипту save.php (в папку data/) const saveBase = fetch('save.php', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify(baseData) }); Одновременно отправляем энциклопедические данные скрипту save_wiki.php (в папку data/) const saveWiki = fetch('save_wiki.php', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify(wikiData) }); Синхронизируем завершение обоих сетевых запросов const [res1, res2] = await Promise.all([saveBase, saveWiki]); Проверяем успешность ответов сервера (HTTP статус 200) if (res1.ok && res2.ok) { window.mainBtn.innerText = «СОХРАНЕНО!»; Выполняем мягкую перезагрузку страницы через 1 секунду для обновления кэша данных setTimeout(() ⇒ location.reload(), 1000);

```
                } else {
                    throw new Error("Сервер ответил отказом при записи файлов.");
                }
            } catch (err) {
                console.error("[Ошибка сохранения баз данных в папку data/]:",
err);
                alert("Не удалось сохранить данные. Проверьте права доступа (CHMOD
777) к папке data.");
        // Возвращаем интерфейс в рабочее состояние для предотвращения потери сессии
                window.mainBtn.innerText = "ОШИБКА";
                window.isLive = true;
            }
        }
    };

}
============================================================
```

========== БЛОК 6: ГЛАВНЫЙ АСИНХРОННЫЙ КОНВЕЙЕР ОБРАБОТКИ РЕЧИ И КОМАНД ========== ========== /* ***Объединенный монолитный обработчик распознанной речи. * Проводит входную строку через каскад фильтров: режимы, перехваты решений, инструменты, лингвистический анализ. * @param {string} text - Распознанная строка текста от голосового движка. */ async function handleSpeech(text) { Предохранитель А: Если Ева говорит в данный момент, полностью игнорируем входящий сигнал if (window.isEvaSpeaking) return; Предохранитель Б: Отсекаем пустые срабатывания микрофона или шумы if (!text || text.trim().length < 1) return; Сбрасываем внутренний таймер инициативы, фиксируя активность пользователя if (typeof resetIdleTimer === 'function') resetIdleTimer(); ПРИНУДИТЕЛЬНЫЙ ВЫВОД: Отображаем фразу пользователя в чате для исключения эффекта зависания if (typeof Visual !== 'undefined' && typeof Visual.addMsg === 'function') { Visual.addMsg(text, 'user-msg'); } Очищаем текстовую плашку предварительного (живого) распознавания речевого потока if (typeof Visual !== 'undefined' && typeof Visual.setLiveText === 'function') { Visual.setLiveText(«»); } Приводим строку к нижнему регистру и очищаем от концевых пробелов для точного сравнения const cleanRaw = text.toLowerCase().trim(); console.log(«[Речевой конвейер]: Начинаю анализ фразы:», cleanRaw); ———————————————————— ШАГ 6.0: ПЕРЕКЛЮЧЕНИЕ РЕЖИМОВ ОБЩЕНИЯ (Мгновенная реакция) ———————————————————— Переход в режим свободной беседы (без постоянного повторения имени «Ева») if (cleanRaw.includes(«давай без имени») || cleanRaw.includes(«давай поболтаем») || cleanRaw.includes(«давай поговорим»)) { window.freeTalk = true; clearTimeout(window.idleTimer); if (Visual.container) Visual.container.classList.add('free-talk-mode'); if (window.mainBtn) window.mainBtn.className = 'btn-free-neon'; const resp = Tools.humanize(window.allData?.settings?.phrases?.freetalk_on || «хорошо, давай поболтаем без имен» || «ладушки, давай без имен»); speak(resp); Visual.addMsg(resp, 'eva-msg'); resetIdleTimer(); return; Прерываем конвейер } Возврат в строгий режим (Ева реагирует только тогда, когда зовут по имени) if (cleanRaw.includes(«хватит болтать») || cleanRaw.includes(«верни имя»)) { window.freeTalk = false; if (Visual.container) Visual.container.classList.remove('free-talk-mode'); if (window.mainBtn) window.mainBtn.className = 'btn-stop-neon'; const resp = Tools.humanize(window.allData?.settings?.phrases?.freetalk_off || «поняла, теперь зови меня по имени»); speak(resp); Visual.addMsg(resp, 'eva-msg'); resetIdleTimer(); return; Прерываем конвейер } ———————————————————— ШАГ 6.1: ПЕРЕХВАТ СОСТОЯНИЯ ОЖИДАНИЯ ВЫБОРА (window.isWaitingDecision) ———————————————————— if (window.isWaitingDecision) { Перехват 1.1: Полный сброс операции и закрытие модальных окон обучения if (cleanRaw.includes(«отмена») || cleanRaw.includes(«проехали») || cleanRaw.includes(«не надо») || cleanRaw === 'выйти' || cleanRaw === 'закрой') { window.isWaitingDecision = false; window.pendingQuestion = «»; window.isSpeakingWiki = false; window.isEvaSpeaking = false; Прячем оранжевые текстовые подсказки управления if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') Visual.updateHints(false); Закрываем любые диалоговые окна обучения через поиск CSS селекторов в DOM const windows = document.querySelectorAll('.learn-window, .learn-form, #learnModal, [id*=«learn»], [class*=«learn»]'); windows.forEach(win ⇒ { win.style.display = 'none'; win.classList.remove('active', 'open', 'show'); }); Возвращаем аватар в режим ожидания и озвучиваем отмену if (typeof Visual !== 'undefined' && typeof Visual.setState === 'function') Visual.setState('wait'); Динамическое считывание фразы отмены из настроек settings.json const resp =***

Tools.humanize(window.allData?.settings?.phrases?.cancel || «хорошо, отменяю» || «ладно, забыли» || «Окей, проехали»); speak(resp); Visual.addMsg(resp, 'eva-msg'); return; }

——— Перехват 1.2: Команда принудительного асинхронного поиска в Wikipedia

——— if (cleanRaw.includes(«поищи») || cleanRaw.includes(«ну поищи») || cleanRaw.includes(«что такое») || cleanRaw.includes(«кто такой») || cleanRaw.includes(«интернет»)) { if (window.pendingQuestion && window.pendingQuestion.length > 0) { window.isWaitingDecision = false; Сбрасываем интерфейсные маркеры и переводим аватар в режим размышления if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') Visual.updateHints(false); if (typeof Visual !== 'undefined' && typeof Visual.setState === 'function') Visual.setState('think'); if (typeof Visual !== 'undefined' && typeof Visual.setLiveText === 'function') { Visual.setLiveText(window.allData?.settings?.phrases?.wiki_searching || «Связываюсь с Википедией…»); } try { let wikiAnswer = null; Каскадный поиск доступного метода интеграции с API Википедии УМНАЯ МОДУЛЬНАЯ МАРШРУТИЗАЦИЯ: Передаем управление модулю инструментов Tools.searchWiki сам заглянет в settings.json и выберет правильную папку навыка! if (typeof Tools !== 'undefined' && typeof Tools.searchWiki === 'function') { wikiAnswer = await Tools.searchWiki(window.pendingQuestion); } else { console.error(«[Речевой конвейер]: Критическая ошибка: Модуль Tools.searchWiki не обнаружен!»); } Если статья была успешно найдена и запущена внутри onWikiResultReceived (Блок 4) if (window.isSpeakingWiki === true || (wikiAnswer && wikiAnswer.trim().length > 5)) { console.log(«[Речевой конвейер]: Вики-запрос успешно обработан в Блоке 4. Подавление дублей сработало.»); Дополнительная подстраховка: если Блок 4 почему-то не вывел текст сам if (window.isSpeakingWiki === false && wikiAnswer && wikiAnswer.trim().length > 5) { speak(wikiAnswer); Visual.addMsg(wikiAnswer, 'eva-msg'); } } else { Сюда система придет только если Википедия действительно сбойнула (ошибка 404 или пусто) window.isLearning = true; if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') { Visual.updateHints(true, «НЕ ЗАПОМИНАЙ / НЕ СОХРАНЯЙ»); } const resp = window.allData?.settings?.phrases?.wiki_not_found || «В интернете не нашла. Научишь меня сам?»; speak(resp); Visual.addMsg(resp, 'eva-msg'); } } catch (e) { ВОССТАНОВЛЕННЫЙ БЛОК ОБРАБОТКИ ОШИБОК: Фиксируем сбой и разблокируем UI console.error(«[Ошибка вызова Wiki API]:», e); window.isSpeakingWiki = false; window.isWaitingDecision = false; } Возвращаем аватар в режим прослушивания if (typeof Visual !== 'undefined' && typeof Visual.setState === 'function') Visual.setState('listen'); } else { speak(«Что именно мне поискать в интернете?»); } return; }

——— Перехват 1.3: Команда ручного обучения текущему вопросу фразами «ответ [текст]»

——— if (cleanRaw.includes(«правильный ответ») || cleanRaw.includes(«ответ») || cleanRaw.includes(«запомни»)) { Вырезаем системные триггеры, оставляя чистый текст ответа пользователя let userAnswer = cleanRaw.replace(/правильный ответ|ответ|запомни/g, «»).trim(); if (userAnswer.length > 1) { Записываем пару Вопрос→Ответ в структуру оперативной памяти ИИ AI.learn(window.pendingQuestion, userAnswer); window.isWaitingDecision = false; if

(typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') { Visual.updateHints(false); } Динамическое считывание фразы успешного обучения из настроек settings.json const resp = Tools.humanize(window.allData?.settings?.phrases?.learned || «Запомнила, теперь я буду это знать!» || «Окей, запомнила» || «Записала!»); speak(resp); if (typeof Visual !== 'undefined' && typeof Visual.addMsg === 'function') { Visual.addMsg(resp, 'eva-msg'); } } else { Если пользователь не назвал сам текст ответа, Ева запрашивает его speak(«Я слушаю, какой правильный ответ?»); } return; Прерываем речевой конвейер на этом шаге }

——— Перехват 1.4: Команда принудительного обращения к локальной нейросети Ollama

——— if (cleanRaw.includes(«подумай») || cleanRaw.includes(«думай») || cleanRaw.includes(«ну подумай») || cleanRaw.includes(«хорошо, подумай») || cleanRaw.includes(«включи мозг») || cleanRaw.includes(«поразмысли»)) { if (window.pendingQuestion && window.pendingQuestion.length > 0) { window.isWaitingDecision = false; Гасим подсказки, включаем режим размышления Евы if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') Visual.updateHints(false); if (typeof Visual !== 'undefined' && typeof Visual.setState === 'function') Visual.setState('think'); if (typeof Visual !== 'undefined' && typeof Visual.setLiveText === 'function') { Visual.setLiveText(«Ева думает…»); } (async () ⇒ { try { Отправляем сохраненный вопрос в ollama.php const response = await fetch('ollama.php', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ message: window.pendingQuestion }) }); const data = await response.json(); let aiResponse = data.message?.content || data.reply || «Не удалось сформулировать ответ.»; if (typeof Tools !== 'undefined' && typeof Tools.humanize === 'function') { aiResponse = Tools.humanize(aiResponse); } Сохраняем диалог в базу знаний ai_base.json через save_ai.php fetch('save_ai.php', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ q: window.pendingQuestion, a: aiResponse }) }) .then(res ⇒ res.json()) .then(saveResult ⇒ console.log(«[База Данных ИИ]:», saveResult.message)) .catch(err ⇒ console.error(«[База Данных ИИ Ошибка записи]:», err)); Озвучка и вывод в чат речевого конвейера if (typeof Visual !== 'undefined' && typeof Visual.setState === 'function') Visual.setState('speak'); speak(aiResponse); if (typeof Visual !== 'undefined' && typeof Visual.addMsg === 'function') { Visual.addMsg(aiResponse, 'eva-msg'); } window.pendingQuestion = «»; Очищаем буфер } catch (error) { console.error(«[Ollama Модуль Критическая ошибка]:», error); window.isWaitingDecision = true; if (typeof Visual !== 'undefined' && typeof Visual.setState === 'function') Visual.setState('listen'); speak(«Ошибка связи с нейросетью. Попробуйте ещё раз.»); } })(); } else { speak(«О чём именно мне подумать?»); } return; } } ——————————————————————— ШАГ 6.2: РЕЖИМ ПРЯМОГО ПОШАГОВОГО ИНТЕРАКТИВНОГО ОБУЧЕНИЯ (window.isLearning) ——————————————————————— if (window.isLearning) { Проверяем команду выхода из жесткого пошагового режима обучения фраз if (cleanRaw.includes(«не запоминай») || cleanRaw.includes(«отмена») || cleanRaw.includes(«проехали»)) { window.isLearning = false; if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') Visual.updateHints(false); Динамическое считывание фразы сброса проехали (forgot) из админки личности Евы speak(Tools.humanize(window.allData?.settings?.phrases?.forgot || «хорошо, проехали»)); return; } Обучаем систему: связываем прошлый зафиксированный вопрос

window.lastQ с текущей репликой ответа AI.learn(window.lastQ, text); window.isLearning = false; if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') Visual.updateHints(false); Динамическое считывание короткой фразы успеха (learned) для пошагового обучения speak(Tools.humanize(window.allData?.settings?.phrases?.learned || «запомнила» || «записала»)); return; } ——————————————————————— ШАГ 6.3: ИНСТРУМЕНТЫ АВТОМАТИЗАЦИИ (МАТЕМАТИКА, ВРЕМЯ, СЛОВАРЬ ХАРАКТЕРА) ——————————————————————— Проверка 3.1: Поиск совпадений в текстовой матрице характера личности Евы из tools.js const personalResp = Tools.processPersonality(cleanRaw); if (personalResp && !window.isWaitingDecision) { speak(personalResp); Visual.addMsg(personalResp, 'eva-msg'); return; } Проверка 3.2: Математический калькулятор (вычисление регулярными выражениями фраз типа «пять плюс два») const mathResult = Tools.calculate(cleanRaw); if (mathResult) { speak(mathResult); Visual.addMsg(mathResult, 'eva-msg'); return; } Проверка 3.3: Запрос текущего системного времени или даты с веб-сервера if (cleanRaw.includes(«время») || cleanRaw.includes(«час») || cleanRaw.includes(«число»)) { const timeStr = await Tools.getServerDateTime(); speak(timeStr); Visual.addMsg(timeStr, 'eva-msg'); return; } ——————————————————————— ШАГ 6.4: ПРОВЕРКА НАЛИЧИЯ ИМЕНИ (АКТИВАЦИОННЫЙ ФИЛЬТР ВХОДНОГО ПОТОКА) ——————————————————————— Система реагирует, если найдено имя «Ева», «Еву» ИЛИ активирован режим freeTalk (без имени) const hasName = cleanRaw.includes(«ева») || cleanRaw.includes(«еву») || window.freeTalk; if (!hasName) { Если имя не произнесено, просто пишем текст в плашку, не запуская глубокий анализ логики баз данных Visual.setLiveText(text); return; } Вычленяем чистый запрос для передачи в ассоциативный ИИ-мозг (ai.js) const cleanForAI = text.replace(/ева|еву/gi, '').trim(); if (cleanForAI.length < 1) return; Выходим, если пользователь назвал только имя Евы без вопроса Рендерим статус размышлений аватара на верхней панели Visual.setState('think'); ——————————————————————— ШАГ 6.5: СЕМАНТИЧЕСКИЙ АССОЦИАТИВНЫЙ ПОИСК В БАЗАХ ЗНАНИЙ (AI.THINK) ——————————————————————— Каскад 1: Ищем в основной ассоциативной базе знаний let aiResult = (typeof AI !== 'undefined' && typeof AI.think === 'function') ? AI.think(cleanForAI) : null; Каскад 2: Если в основной базе пусто — ищем в базе ИИ (ai_base.json) if ((!aiResult || !aiResult.found) && window.allData?.aiBase) {

```
   // Пробегаем по массиву вопросов ai_base.json, приводя строки к нижнему регистру
      const aiBaseMatch = window.allData.aiBase.find(item =>
          item.questions && item.questions.some(q => q.toLowerCase().trim()
=== cleanForAI.toLowerCase())
      );
```

```
      if (aiBaseMatch) {
          console.log("[Ева]: Ответ успешно найден в ai_base.json");
          aiResult = {
              found: { answers: aiBaseMatch.answers || ["По памяти."] },
              clean: cleanForAI
          };
      }
  }
```

```
 // Третий каскад: Если и в ai_base пусто — ищем в энциклопедической Wiki-базе
(wiki_base.json)
```

```
  if ((!aiResult || !aiResult.found) && window.allData?.wiki) {
   // Пробегаем по массиву вопросов wiki-базы, приводя строки к нижнему регистру
      const wikiMatch = window.allData.wiki.find(item =>
          item.questions && item.questions.some(q => q.toLowerCase().trim()
=== cleanForAI.toLowerCase())
      );
      if (wikiMatch) {
     // ЗАЩИТНЫЙ МЕХАНИЗМ: Формируем искусственный массив answers, если структура
Wiki-объекта отличается
          console.log("[Ева]: Ответ успешно найден в wiki_base.json");
          const fallbackAnswers = wikiMatch.answers || [wikiMatch.text ||
wikiMatch.description || "По записям."];
          aiResult = {
              found: { answers: fallbackAnswers || ["По записям."] },
              clean: cleanForAI
          };
      }
  }
```

```
 // ВЫВОД РЕЗУЛЬТАТОВ АНАЛИЗА БАЗ ДАННЫХ
  if (aiResult && aiResult.found && aiResult.found.answers &&
aiResult.found.answers.length > 0) {
   // Реакция А: Ответ НАЙДЕН. Выбираем случайный вариант из массива ответов для
живости диалога
      let response = aiResult.found.answers[Math.floor(Math.random() *
aiResult.found.answers.length)];
```

```
   // Прогоняем сухой ответ через фильтр лингвистического очеловечивания и обращения по
имени
      if (typeof Tools !== 'undefined' && typeof Tools.humanize ===
'function') {
          response = Tools.humanize(response);
      }
```

```
   // Переводим аватар в статус генерации реплики
      Visual.setState('speak');
      speak(response);
      Visual.addMsg(response, 'eva-msg');
   // Фиксируем контекст текущей ноды для обеспечения поддержки древовидных sub-
веток
      if (typeof AI !== 'undefined') AI.currentContext = aiResult.found;
  } else {
      //
==========================================================================
   // □ РЕАКЦИЯ Б: ОТВЕТ АБСОЛЮТНО НЕ НАЙДЕН В ПАМЯТИ ИИ ЕВЫ
   // Активируем шлюзы интерактивного диалога обучения и ожидания решения
      //
==========================================================================
      window.isWaitingDecision = true;
      window.pendingQuestion   = cleanForAI;
   // Запоминаем текущий чистый вопрос для режима прямого пошагового обучения
```

```
      window.lastQ = cleanForAI;
   // Считываем строку оранжевых подсказок выбора действий из папки
data/settings.json
      const currentDecisionHints =
window.allData?.settings?.hints?.decision_hints || "ПОИЩИ / ПОДУМАЙ / ОТВЕТ...
/ ОТМЕНА";
   // Обновляем оранжевые подсказки действий на верхнем статус-баре экрана Евы
      if (typeof Visual !== 'undefined' && typeof Visual.updateHints ===
'function') {
          Visual.updateHints(true, currentDecisionHints);
      }
   // Считываем главный предустановленный вопрос Евы из папки data/settings.json
      const prompt = window.allData?.settings?.phrases?.no_answer || "Я не
знаю ответа, но могу подумать! А может подскажешь ответ? Или мне поискать в интернете?";
   // Запускаем генерацию речи ИИ через TTS речевую систему voice_system.js
      speak(prompt);
   // Выводим текстовое облако сообщения ассистента в центральный чат приложения
      if (typeof Visual !== 'undefined' && typeof Visual.addMsg ===
'function') {
          Visual.addMsg(prompt, 'eva-msg');
      }
  }
```

}
==========
========== БЛОК 7: АВТОНОМНЫЙ ТАЙМЕР ИНИЦИАТИВЫ И ПРОАКТИВНОСТИ ЕВЫ (СКУКА)
==========
========== */ **** Сброс и перезапуск таймера неактивности пользователя. * Если пользователь молчит, Ева с вероятностью 30% может проявить инициативу и задать вопрос сама. */ function resetIdleTimer() { Стираем предыдущий запущенный таймер активности clearTimeout(window.idleTimer); Защитные условия: Ева молчит, если микрофон выключен, идет обучение или ожидается решение команды if (!window.isLive*** || ***window.isLearning*** || ***window.isWaitingDecision) return; Запускаем новый таймер ожидания на базе глобальной конфигурации window.IDLE_TIME window.idleTimer = setTimeout(() ⇒ { Проверяем, не говорит ли Ева прямо сейчас, и вычисляем случайный шанс (30%) if (!window.speechSynthesis.speaking && Math.random() ⇐ 0.3) { Запрашиваем из ИИ случайную заготовку вопроса для пользователя const entry = AI.getEvaQuestionEntry(); if (entry) { Выбираем случайный вариант вопроса, очеловечиваем его и воспроизводим const text = Tools.humanize(entry.answers[Math.floor(Math.random() * entry.answers.length)]); AI.currentContext = entry; Запоминаем контекст для анализа ответа пользователя speak(text); Visual.addMsg(text, 'eva-msg'); } Рекурсивно перезапускаем таймер для следующего шага проверки resetIdleTimer(); } else { Если условия рандома не прошли, просто уходим на новый круг ожидания resetIdleTimer(); } }, window.IDLE_TIME); }***
**==========
========== *БЛОК 8: СЕРВЕРНОЕ АВТОМАТИЧЕСКОЕ СОХРАНЕНИЕ ДАННЫХ И СБРОС БЛОКИРОВОК***
==========
========== */ ** Асинхронное автоматическое сохранение полученных из интернета*

*знаний на server. * Выполняет обновление оперативной памяти, запись в файл и полную разблокировку UI. * @param {string} question - Вопрос пользователя. * @param {string} answer - Текст найденной статьи из Wikipedia. */ async function autoSaveToWiki(question, answer) { console.log(«[AutoSave Модуль]: Подготовка структуры объекта данных для записи...»);*
Создаем каноничную структуру сущности знаний для базы данных

```
  const newEntry = {
      type: "qa",
      questions: [question.toLowerCase().trim()],
      answers: [answer],
      sub: []
  };
```

```
 // Проверяем существование массива и пушим новый элемент в оперативную память браузера
  if (!window.allData.wiki) window.allData.wiki = [];
  window.allData.wiki.push(newEntry);
```

```
  console.log("[AutoSave  Модуль]: Отправка обновленного массива в Database.saveWiki...");
 // Асинхронно передаем массив модулю сохранения и дожидаемся ответа от сервера
  const success = await Database.saveWiki(window.allData.wiki);
 // Анализируем маркер успешности проведения дисковой операции записи сервером
  if (success) {
      console.log("%c[AutoSave  Модуль]: УСПЕШНО ЗАПИСАНО В  wiki_base.json", "color: green; font-weight: bold;");
   // --- КРИТИЧЕСКИЙ СБРОС ВСЕХ СИСТЕМНЫХ БЛОКИРОВОК ДЛЯ ПОДДЕРЖАНИЯ ДИАЛОГА ---
      window.isWaitingDecision = false; //  Разблокируем шлюзы обработки новых фраз (Слон, Бублик)
      window.pendingQuestion   = "";     //  Очищаем буфер временного вопроса
      window.isSpeakingWiki    = false;   //  Отключаем защиту чтения статьи Википедии
      window.isEvaSpeaking     = false;    //  Сбрасываем статус речи
```

```
   // --- ИНТЕРФЕЙСНАЯ ОЧИСТКА ЭКРАНА ---
   // Полностью очищаем текстовую строку от застрявших серых/мусорных букв
      if (typeof Visual !== 'undefined' && typeof Visual.setLiveText === 'function') {
          Visual.setLiveText("");
      }
```

```
   // Деактивируем оранжевую строку подсказок команд
      if (typeof Visual !== 'undefined' && typeof Visual.updateHints === 'function') {
          Visual.updateHints(false);
      }
```

```
   // Автоматически закрываем всплывающие формы обучения диалогов изменением инлайновых  CSS  свойств
      const windows = document.querySelectorAll('.learn-window, .learn-form,
```

```
#learnModal, [id*="learn"], [class*="learn"]');
      windows.forEach(win => {
          win.style.display = 'none';
          win.classList.remove('active', 'open', 'show');
      });
```

```
      if (typeof Visual !== 'undefined' && typeof Visual.setState === 
'function') {
          Visual.setState('wait');
      }
```

```
   // --- ПРОАКТИВНАЯ АКТИВАЦИЯ МИКРОФОНА ---
   // Если сессия продолжается, принудительно перезапускаем аудиопоток захвата голоса
      if (window.isLive && window.rec) {
          try {
              window.rec.start();
              console.log("%c[AutoSave  Модуль]: МИКРОФОН СНОВА АКТИВЕН ДЛЯ 
НОВОГО ДИАЛОГА!", "color: green; font-weight: bold;");
          } catch(e) {
              console.log("[AutoSave  Модуль]: Поток микрофона уже запущен или 
управляется извне.");
          }
      }
  } else {
      console.error("[AutoSave  Модуль]: КРИТИЧЕСКАЯ ОШИБКА: Сервер вернул статус 
false  при записи файла.");
  }
```

```
}
================================================================
============== БЛОК 9: ТОЧКА ВХОДА И РЕГИСТРАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ
================================================================
============== Автоматический выпуск первичного каскада инициализации при подключении скрипта startApp(); *ГЛОБАЛЬНАЯ РЕГИСТРАЦИЯ: Назначаем финальную монолитную функцию handleSpeech в глобальное окно window window.handleSpeech = handleSpeech;*
```