Codebase Review Report: EduLearn Recent Changes

Report Date: November 29, 2025
Reviewer: System Optimization Analyst
Focus: Recent uncommitted changes and architectural decisions

---

1. Summary of Key Observations

Breaking Changes and Critical Issues

> [!WARNING] **Dual AI Implementation Architecture Detected** - The codebase now has TWO separate implementations for AI chat functionality, creating architectural redundancy and potential maintenance complexity.

> [!IMPORTANT] **Hardcoded API Key in Version Control** - The `.env.local` file contains a Google Generative AI API key that appears to be tracked in git, creating a security vulnerability.

Major Modifications Detected

Uncommitted Changes:

- 6 modified files (2,122 insertions, 140 deletions)
- 3 new untracked directories/files
- Significant chatbot infrastructure expansion

Key Areas of Change:

1. **Backend API Enhancement** - 4 new REST endpoints for chat session management ([chatbot.ts](file:///Users/naveen/Documents/GitHub/EduLearn/backend/src/routes/chatbot.ts))
2. **Next.js API Route** - New AI streaming endpoint bypassing backend ([route.ts](file:///Users/naveen/Documents/GitHub/EduLearn/app/api/chat/route.ts))
3. **Neuro-Adaptive AI System** - Learning style personalization framework ([neuro-prompts.ts](file:///Users/naveen/Documents/GitHub/EduLearn/lib/neuro-prompts.ts))
4. **Frontend Widget Overhaul** - Chat history UI, markdown rendering, session management
5. **Authentication Extension** - Learning style field added to user type
6. **Package Dependencies** - 4 new AI/markdown libraries added

---

2. Detailed Change Analysis

Change #1: Dual AI Architecture Implementation

Original Problem

The system needed enhanced chat capabilities with session persistence, markdown rendering, and better streaming support.

What Changed

Architecture Decision: Created two parallel AI implementations:

1. **Backend Webhook Path (Original - Modified)**
   - Location: [backend/src/routes/chatbot.ts] (file:///Users/naveen/Documents/GitHub/EduLearn/backend/src/routes/chatbot.ts)
   - External webhook integration via `EXTERNAL_WEBHOOK_URL`
   - Database logging of sessions and messages
   - Single `/query` endpoint

2. **Next.js API Route Path (New)**
   - Location: [app/api/chat/route.ts] (file:///Users/naveen/Documents/GitHub/EduLearn/app/api/chat/route.ts)
   - Direct Google Gemini 2.0 Flash integration
   - Client-side streaming with `ReadableStream`
   - Bypasses backend entirely for AI responses

Why This Change Was Made

Based on conversation history, the original webhook-based system likely had streaming errors and reliability issues. The new Next.js route provides:

- Direct control over AI streaming
- Better error handling for chunk parsing
- Faster response times (no webhook round-trip)
- Runtime flexibility (Node.js runtime explicitly set)

Expected Behavior

- Frontend can choose between webhook-based chat (via backend) or direct AI streaming (via Next.js API)
- Better markdown rendering with `react-markdown` and `remark-gfm`
- Session persistence with chat history sidebar
- Learning style-adapted responses

Change #2: Chat Session Management API

Original Problem

No backend support for retrieving, listing, or managing historical chat sessions.

What Changed

Added 4 new endpoints to [chatbot.ts] (file:///Users/naveen/Documents/GitHub/EduLearn/backend/src/routes/chatbot.ts):

```
POST   /api/chatbot/log              // Log messages from frontend
GET    /api/chatbot/sessions         // List all user sessions
```

```
GET    /api/chatbot/sessions/:id     // Get specific session messages
DELETE /api/chatbot/sessions/:id     // Soft-delete session
```

Database Schema:

- `chat_sessions` table: Stores session metadata, timestamps, soft-delete support
- `chat_messages` table: Stores individual messages with sender type
- 6 indexes for performance optimization

Why Necessary

Without this API layer, the frontend couldn't display chat history or manage sessions. This enables:

- Persistent conversation retrieval
- Session deletion (privacy compliance)
- User-specific chat isolation via JWT authentication

Expected Behavior

- Users can view all previous chat sessions
- Click to load historical conversations
- Delete unwanted sessions (soft delete in DB)
- Metadata tracking (started_at, last_activity, summary)

Change #3: Neuro-Adaptive Prompt System

Original Addition

New file: [lib/neuro-prompts.ts](file:///Users/naveen/Documents/GitHub/EduLearn/lib/neuro-prompts.ts)

What It Does

Creates personalized AI system prompts based on:

Learning Styles:

- **ADHD**: Short paragraphs, bullet points, bold key terms, energetic tone
- **Dyslexia**: Double spacing, simple language, TL;DR summaries
- **Anxiety**: Supportive tone, tiny manageable steps, validation
- **General**: Standard clear markdown

User Roles:

- Student: Focus on learning, understanding
- Teacher: Lesson planning, rubrics, classroom management
- Admin: System administration, data analysis
- Parent: Child's progress, curriculum support

Global Guardrail:

```
CRITICAL INSTRUCTION: You are an AI tutor, NOT a homework machine.
- NEVER provide direct answers to homework questions
- ALWAYS guide through leading questions
- Maintain academic integrity
```

Why This Matters

Demonstrates commitment to accessibility and personalized learning experiences. Addresses diverse cognitive needs.

Integration

- Used in [app/api/chat/route.ts] (file:///Users/naveen/Documents/GitHub/EduLearn/app/api/chat/route.ts#L49) via `generateSystemPrompt()`
- Leverages Gemini's `systemInstruction` parameter
- `learningStyle` field added to [lib/types/auth.ts] (file:///Users/naveen/Documents/GitHub/EduLearn/lib/types/auth.ts#L9)

Change #4: Frontend Widget Transformation

Original State

Basic chat interface with simple text rendering.

What Changed

Major Enhancements to [chatbot-widget.tsx] (file:///Users/naveen/Documents/GitHub/EduLearn/components/chatbot/chatbot-widget.tsx):

1. **Chat History Sidebar**
    - Fetches sessions via `GET /api/chatbot/sessions`
    - Displays session summaries with timestamps
    - Load previous conversations on click
    - Delete button with confirmation
2. **Markdown Rendering**
    - Integrated `ReactMarkdown` with `remark-gfm`
    - Custom component styling for code blocks, tables, lists
    - Syntax highlighting for code fences
    - Blockquote, heading, and link formatting
3. **Enhanced File Upload**
    - Added image support: `.jpg`, `.jpeg`, `.png`, `.webp`
    - Previously: Only documents (pdf, doc, xls, ppt, txt, csv)

4. **UX Improvements**
    - Enter key to send (Shift+Enter for newline)
    - Disabled state when not authenticated
    - Loading state with "Thinking..." placeholder
    - Improved button states with transitions

Technical Details

Streaming Integration:

```
const response = await fetch('/api/chat', {
  method: 'POST',
  body: JSON.stringify({ messages, userProfile, sessionId })
});
const reader = response.body?.getReader();
// Stream chunks and accumulate response
```

Session State Management:

- `sessionId`: UUID v4 generated on component mount
- `chatHistory`: Array of session objects
- `loadSession()`: Fetches messages and updates UI

Change #5: Package Dependencies

New Dependencies Added

Package	Version	Purpose
`@google/generative-ai`	^0.24.1	Google Gemini AI SDK
`ai`	^5.0.104	Vercel AI SDK (streaming utilities)
`react-markdown`	^10.1.0	Markdown rendering in React
`remark-gfm`	^4.0.1	GitHub Flavored Markdown support

Bundle Impact:

- Significant size increase (~1600 lines in package-lock.json)
- Multiple sub-dependencies for markdown parsing

Why Each Library

- **@google/generative-ai**: Direct Gemini integration without webhook
- **ai**: Edge runtime streaming helpers (though not visibly used)
- **react-markdown**: Professional markdown display for AI responses
- **remark-gfm**: Tables, strikethrough, task lists support

Change #6: Authentication Type Extension

Minimal but Strategic

Added to [lib/types/auth.ts](file:///Users/naveen/Documents/GitHub/EduLearn/lib/types/auth.ts):

```
export interface User {
  // ... existing fields
  learningStyle?: 'ADHD' | 'Dyslexia' | 'Anxiety' | 'General';
}
```

Impact:

- Optional field, backward compatible
- Enables neuro-adaptive prompt selection
- Requires frontend user profile management (not visible in diffs)

---

3. Drawbacks and Limitations

🔴 Critical Issues

1. Architectural Redundancy: Dual AI Paths

Problem:
Two separate AI chat implementations exist simultaneously:

- Backend webhook route (`/api/chatbot/query`)
- Next.js API route (`/app/api/chat/route.ts`)

Risks:

- **Code Duplication**: Database logging logic duplicated in both paths
- **Maintenance Burden**: Bug fixes must be applied twice
- **Decision Ambiguity**: Unclear which path the frontend actually uses
- **Performance Waste**: Backend infrastructure maintained but potentially unused

Evidence:

- Frontend widget makes fetch to `/api/chat` (Next.js route)
- Backend `/chatbot/query` webhook path still fully implemented
- No deprecation markers or migration plan visible

Recommendation: Choose one implementation and deprecate the other.

2. Hardcoded Secrets in Version Control

Problem:
`.env.local` file modified in git with production API key:

```
GOOGLE_GENERATIVE_AI_API_KEY=AIzaSyC0FClC4s8Zzq6ceb3ilAV7D1ExH2TgxNE
```

Security Impact:

- API key visible in git diffs
- Potential exposure if repository becomes public
- Key rotation requires code changes
- Violates security best practices

Immediate Actions Required:

1. Rotate the exposed API key
2. Add `.env.local` to `.gitignore`
3. Remove from git history: `git rm --cached .env.local`
4. Use environment-specific secrets management

3. Missing Database Migration for `learningStyle` Field

Problem:
Code references `user.learningStyle` but database schema shows:

```
CREATE TABLE users (
  role user_role NOT NULL,
  -- no learningStyle column
)
```

Impact:

- Runtime errors when fetching user profiles
- Neuro-adaptive prompts cannot function
- Frontend may crash on user data access

Required Migration:

```
ALTER TABLE users ADD COLUMN learning_style VARCHAR(20)
  CHECK (learning_style IN ('ADHD', 'Dyslexia', 'Anxiety', 'General'));
CREATE INDEX idx_users_learning_style ON users(learning_style);
```

4. Uncontrolled AI Streaming Costs

Problem:
Next.js route directly calls Gemini API with:

- No rate limiting
- No quota management

- `maxOutputTokens: 4000` (could be expensive at scale)
- Unauthenticated requests possible (only backend authenticates)

Cost Projection:

- Gemini 2.0 Flash: ~$0.075 per 1M input tokens, ~$0.30 per 1M output
- 1000 daily users × 10 messages × 4000 tokens = 40M tokens/day
- Potential cost: $12/day = $360/month (output only)

Missing Safeguards:

- No per-user request throttling
- No monthly API budget alerts
- No fallback to cached responses

⚠ Significant Issues

5. Session Management State Inconsistency

Problem:
Frontend generates `sessionId` on component mount:

```
const [sessionId] = useState(() => crypto.randomUUID());
```

Flaws:

- New UUID on every page refresh → lost session continuity
- Not persisted to localStorage → no cross-tab consistency
- Logging happens async, may not complete before page unload
- Race condition: User message logged before session created

Better Approach:

```
const [sessionId] = useState(() => {
  const stored = localStorage.getItem('activeSessionId');
  return stored || crypto.randomUUID();
});
```

6. No Error Recovery in Streaming

Problem:
Stream error handling just logs and closes:

```
catch (e) {
  console.error('Stream error:', e);
  controller.error(e);
}
```

Missing:

- User-facing error message
- Retry mechanism with exponential backoff
- Fallback to non-streaming response
- Partial response recovery

User Experience:

- Silent failures leave user confused
- No indication of what went wrong
- Must manually refresh to retry

7. Index Coverage Gaps in Database

Current Indexes:

```
CREATE INDEX idx_chat_sessions_user_id ON chat_sessions(user_id);
CREATE INDEX idx_chat_sessions_deleted ON chat_sessions(is_deleted);
```

Missing Important Index:

```
-- Query: "WHERE user_id = ? AND is_deleted = false ORDER BY last_activity
DESC"
CREATE INDEX idx_chat_sessions_user_active
  ON chat_sessions(user_id, is_deleted, last_activity DESC);
```

Performance Impact:

- Index scan on user_id, then filter by is_deleted
- Sorting happens after filtering (not index-optimized)
- Slower queries as session count grows

8. Webhook Dependency Without Fallback

Problem:
Backend route relies on `EXTERNAL_WEBHOOK_URL`:

```
if (!webhookUrl) {
  return res.status(500).json({ error: 'Webhook not configured' });
}
```

Risks:

- Single point of failure (N8N or external service down = total outage)
- No health check before request
- No timeout configuration (default fetch timeout may be too long)
- No circuit breaker pattern

Better Architecture:

```
const timeout = 10000; // 10s
const controller = new AbortController();
setTimeout(() => controller.abort(), timeout);

try {
  const response = await fetch(webhookUrl, {
    signal: controller.signal,
    // request config
  });
} catch (error) {
  if (error.name === 'AbortError') {
    // Log timeout, use fallback
  }
}
```

9. Massive File Uploads Allowed Without Validation

Problem:
File upload accepts multiple files without size limits:

```
<input type="file" multiple accept="..." />
```

Missing Validations:

- No maximum file size check
- No total batch size limit
- No file count restriction
- Images added without compression

Attack Vectors:

- Upload 100× 50MB images = 5GB request
- Exhaust server memory
- DoS via large file processing

Needed Checks:

```
const MAX_FILE_SIZE = 10 * 1024 * 1024; // 10MB
const MAX_FILES = 5;
const MAX_TOTAL_SIZE = 25 * 1024 * 1024; // 25MB
```

```
if (files.length > MAX_FILES) {
  alert(`Maximum ${MAX_FILES} files allowed`);
  return;
}
```

10. No Markdown Sanitization

Problem:
Markdown rendered directly from AI without sanitization:

```
<ReactMarkdown>{message.text}</ReactMarkdown>
```

Risks:

- AI could be manipulated to inject malicious markdown
- No XSS protection (though ReactMarkdown has some default safety)
- Links open with `target="_blank"` but missing `rel="noopener"`

Already Mitigated Partially:

```
// Code has this
<a target="_blank" rel="noopener noreferrer" />
```

But no explicit HTML sanitization library like `DOMPurify`.

📊 Performance and Scalability Concerns

11. Database Query Pattern Anti-pattern

Current Implementation:

```
// First check session ownership
const sessionCheck = await query(`
  SELECT id FROM chat_sessions WHERE session_token = $1 AND user_id = $2
`, [sessionId, userId]);

// Then fetch messages
const result = await query(`
  SELECT * FROM chat_messages WHERE session_id = $1
`, [sessionCheck.rows[0].id]);
```

Problem:
Two sequential queries when one would suffice.

Optimized Version:

```
const result = await query(`
  SELECT cm.*
  FROM chat_messages cm
  JOIN chat_sessions cs ON cm.session_id = cs.id
  WHERE cs.session_token = $1
    AND cs.user_id = $2
    AND cm.is_deleted = false
  ORDER BY cm.created_at ASC
`, [sessionId, userId]);
```

Impact:

- 50% reduction in database round trips
- Better performance under high concurrency
- Atomic operation (no race conditions)

12. Frontend Re-renders on Every Stream Chunk

Problem:
Message state updated on every chunk:

```
for await (const chunk of stream) {
  setMessages(prev => {
    const updated = [...prev];
    updated[updated.length - 1].text += chunk;
    return updated;
  });
}
```

React Performance Impact:

- Re-render on every chunk (could be 100+ per response)
- Entire message list re-rendered (not just last message)
- Markdown re-parsing on every chunk

Optimization:

```
// Use ref for accumulation, setState once at end
const textRef = useRef('');
for await (const chunk of stream) {
  textRef.current += chunk;
  // Update UI via ref, not state
}
setMessages(prev => [...prev, { text: textRef.current }]);
```

13. No Database Connection Pooling Configuration Visible

Problem:
Database import shows:

```
import { query } from '../config/db';
```

But no evidence of pool size tuning for chat workload.

Chat-Specific Needs:

- High read:write ratio (loading history >> sending messages)
- Concurrent users polling for sessions
- Long-lived connections for streaming

Recommended Config:

```
const pool = new Pool({
  max: 20,              // Up from default 10
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 3000,
  // Read replicas for SELECT queries
});
```

14. Low Bandwidth Performance Not Addressed

As Per User Request: Evaluate low-bandwidth scalability.

Current Issues:

1. **Markdown Libraries Bundle Size**
    - `react-markdown` + `remark-gfm` ≈ 200KB uncompressed
    - No code splitting or lazy loading
    - Loaded even if user doesn't use chat
2. **No Response Compression**
    - Streaming endpoint returns plain text
    - Missing: `Content-Encoding: gzip` header
    - 4000 token response ≈ 16KB (could be 4KB compressed)
3. **Chat History Fetches All Messages**
    - No pagination: `SELECT * FROM chat_messages`
    - User with 1000 messages in session = huge JSON payload
    - Should implement cursor-based pagination

4. **Base64 Image Encoding for Attachments**
    - Images encoded as base64 in JSON
    - 33% size overhead vs binary
    - Better: Upload to S3, send URL reference

Low Bandwidth Recommendations:

```
// Paginated messages
GET /api/chatbot/sessions/:id?limit=50&cursor=<messageId>

// Progressive markdown loading
import dynamic from 'next/dynamic';
const ReactMarkdown = dynamic(() => import('react-markdown'), {
  loading: () => <p>Loading...</p>,
  ssr: false
});
```

---

4. Recommendations & Improvements

🚀 Short-Term Fixes (1-2 Weeks)

1. Resolve Architectural Redundancy

Action Plan:

1. **Decision Matrix:**
    - If webhook provides specialized logic (content filtering, analytics): Keep backend path, remove Next.js route
    - If direct AI control needed: Keep Next.js route, deprecate backend webhook
2. **Migration Steps:**

```
// Option A: Deprecate Next.js route
export async function POST(req: Request) {
  return new Response(JSON.stringify({
    error: 'This endpoint is deprecated. Use /api/chatbot/query'
  }), { status: 410 });
}

// Option B: Deprecate webhook
router.post('/query', (req, res) => {
  res.status(410).json({
    error: 'Webhook path deprecated. Frontend uses direct AI
streaming.'
  });
});
```

3. **Code Cleanup:**
    - Remove unused route files
    - Update frontend to single path
    - Remove deprecated packages if unused

2. Security Hardening

Immediate Actions:

```
# 1. Remove .env.local from git
git rm --cached .env.local
echo ".env.local" >> .gitignore

# 2. Rotate API key at Google Cloud Console

# 3. Remove from history (nuclear option)
git filter-branch --force --index-filter \
  "git rm --cached --ignore-unmatch .env.local" \
  --prune-empty --tag-name-filter cat -- --all
```

Environment Secret Management:

```
// For production
// Use Vercel environment variables (encrypted at rest)
// Or AWS Secrets Manager, HashiCorp Vault

// For local development
// Create .env.local from template
cp .env.local.template .env.local
# Add to .gitignore (already done)
```

3. Add Database Migration

migration_add_learning_style.sql:

```
-- Add learning_style column to users table
ALTER TABLE users
ADD COLUMN learning_style VARCHAR(20)
  CHECK (learning_style IN ('ADHD', 'Dyslexia', 'Anxiety', 'General'));

-- Add index for query performance
CREATE INDEX idx_users_learning_style ON users(learning_style)
  WHERE learning_style IS NOT NULL;

-- Set default for existing users
UPDATE users SET learning_style = 'General' WHERE learning_style IS NULL;
```

```
-- Make it optional but indexed
ALTER TABLE users ALTER COLUMN learning_style DROP NOT NULL;
```

Integration:

```
// backend/src/migrations/
import { query } from '../config/db';
export async function up() {
  await query(/* SQL above */);
}
```

4. Implement Request Rate Limiting

For Next.js API Route:

```
// app/api/chat/rate-limit.ts
import { Ratelimit } from '@upstash/ratelimit';
import { Redis } from '@upstash/redis';

const ratelimit = new Ratelimit({
  redis: Redis.fromEnv(),
  limiter: Ratelimit.slidingWindow(10, '1 m'), // 10 requests per minute
  analytics: true,
});

export async function checkRateLimit(userId: string) {
  const { success, limit, remaining } = await ratelimit.limit(userId);
  return { allowed: success, limit, remaining };
}

// In route.ts
const { allowed } = await checkRateLimit(userId);
if (!allowed) {
  return new Response('Rate limit exceeded', { status: 429 });
}
```

Cost Protection:

```
// Monthly quota tracking
const MONTHLY_TOKEN_LIMIT = 10_000_000; // 10M tokens
const currentUsage = await redis.get(`usage:${month}`);
if (currentUsage > MONTHLY_TOKEN_LIMIT) {
  // Fallback to cached responses or error
}
```

5. Fix Session Persistence

Updated Widget Logic:

```
// chatbot-widget.tsx
const [sessionId, setSessionId] = useState(() => {
  if (typeof window === 'undefined') return '';

  const STORAGE_KEY = 'edulearn_chat_session';
  const stored = localStorage.getItem(STORAGE_KEY);

  if (stored) {
    try {
      const { id, timestamp } = JSON.parse(stored);
      const age = Date.now() - timestamp;
      // Session valid for 24 hours
      if (age < 24 * 60 * 60 * 1000) return id;
    } catch {}
  }

  const newId = crypto.randomUUID();
  localStorage.setItem(STORAGE_KEY, JSON.stringify({
    id: newId,
    timestamp: Date.now()
  }));
  return newId;
});

// New session button
const startNewSession = () => {
  const newId = crypto.randomUUID();
  localStorage.setItem(STORAGE_KEY, JSON.stringify({
    id: newId,
    timestamp: Date.now()
  }));
  setSessionId(newId);
  setMessages([]);
};
```

6. Add Stream Error Recovery

Enhanced Error Handling:

```
// app/api/chat/route.ts
async start(controller) {
  const encoder = new TextEncoder();
  let fullResponse = '';
  let retryCount = 0;
  const MAX_RETRIES = 2;
```

```
    const attemptStream = async () => {
      try {
        for await (const chunk of result.stream) {
          const text = chunk.text();
          if (text) {
            fullResponse += text;
            controller.enqueue(encoder.encode(text));
          }
        }
        return true;
      } catch (e) {
        console.error(`Stream error (attempt ${retryCount + 1}):`, e);

        if (retryCount < MAX_RETRIES && isRetryable(e)) {
          retryCount++;
          await sleep(1000 * retryCount); // Exponential backoff
          return attemptStream();
        }

        // Send error message to user
        const errorMsg = '\n\n_[An error occurred. Please try again.]_';
        controller.enqueue(encoder.encode(errorMsg));
        throw e;
      }
    };

    await attemptStream();
    controller.close();
  }

  function isRetryable(error: any): boolean {
    return error.message?.includes('network') ||
           error.status === 503 ||
           error.status === 429;
  }
```

🏗️ Long-Term Architectural Improvements (1-3 Months)

7. Implement Caching Layer

Problem: Every AI request hits Gemini API (costly and slow).

Solution: Redis-based cache for common queries

```
// lib/cache/chat-cache.ts
import { Redis } from '@upstash/redis';

const redis = Redis.fromEnv();

interface CacheEntry {
  response: string;
```

```
  timestamp: number;
  hits: number;
}

export async function getCachedResponse(
  query: string,
  userProfile: NeuroProfile
): Promise<string | null> {
  const key =
`chat:${hash(query)}:${userProfile.role}:${userProfile.learningStyle}`;
  const cached = await redis.get<CacheEntry>(key);

  if (cached && Date.now() - cached.timestamp < 7 * 24 * 60 * 60 * 1000) {
    // Update hit count
    await redis.set(key, { ...cached, hits: cached.hits + 1 });
    return cached.response;
  }

  return null;
}

export async function setCachedResponse(
  query: string,
  userProfile: NeuroProfile,
  response: string
) {
  const key =
`chat:${hash(query)}:${userProfile.role}:${userProfile.learningStyle}`;
  await redis.set(key, {
    response,
    timestamp: Date.now(),
    hits: 1
  }, { ex: 7 * 24 * 60 * 60 }); // 7 day TTL
}

// In route.ts
const cached = await getCachedResponse(lastMessage.content, userProfile);
if (cached) {
  return new Response(cached);
}
// ... else call AI and cache result
```

Expected Impact:

- 30-50% reduction in API costs for common questions
- 10x faster response for cached queries
- Improved low-bandwidth performance

8. Message Pagination Strategy

Current: Fetch all messages for session
Target: Cursor-based pagination

API Changes:

```
// GET /api/chatbot/sessions/:sessionId/messages
interface MessageQuery {
  limit?: number;      // Default 50
  cursor?: string;     // Message ID
  direction?: 'before' | 'after';
}

router.get('/sessions/:sessionId/messages', async (req, res) => {
  const { limit = 50, cursor, direction = 'before' } = req.query;

  const query = `
    SELECT cm.*
    FROM chat_messages cm
    JOIN chat_sessions cs ON cm.session_id = cs.id
    WHERE cs.session_token = $1 AND cs.user_id = $2
      AND cm.is_deleted = false
      ${cursor ? `AND cm.created_at ${direction === 'before' ? '<' : '>'}
(
        SELECT created_at FROM chat_messages WHERE id = $3
      )` : ''}
    ORDER BY cm.created_at ${direction === 'before' ? 'DESC' : 'ASC'}
    LIMIT $${cursor ? 4 : 3}
  `;

  const params = cursor ? [sessionId, userId, cursor, limit] : [sessionId,
userId, limit];
  const result = await query(query, params);

  res.json({
    messages: result.rows,
    hasMore: result.rows.length === limit,
    nextCursor: result.rows[result.rows.length - 1]?.id
  });
});
```

Frontend Integration:

```
// Infinite scroll in chat history
const loadMoreMessages = async () => {
  if (!nextCursor) return;

  const res = await fetch(
    `/api/chatbot/sessions/${sessionId}/messages?
cursor=${nextCursor}&limit=50`
  );
  const { messages, hasMore, nextCursor: newCursor } = await res.json();

  setMessages(prev => [...messages, ...prev]);
```

```
  setNextCursor(newCursor);
  setHasMore(hasMore);
};
```

9. Observability and Monitoring

Implement Comprehensive Logging:

```
// lib/monitoring/chat-telemetry.ts
import * as Sentry from '@sentry/nextjs';
import { PostHog } from 'posthog-node';

export interface ChatMetrics {
  sessionId: string;
  userId: string;
  requestTokens: number;
  responseTokens: number;
  latencyMs: number;
  cached: boolean;
  learningStyle: string;
  error?: string;
}

export async function trackChatMetrics(metrics: ChatMetrics) {
  // PostHog for analytics
  posthog.capture({
    distinctId: metrics.userId,
    event: 'chat_request',
    properties: metrics
  });

  // Sentry for performance monitoring
  const transaction = Sentry.startTransaction({
    name: 'chat.request',
    op: 'ai.inference',
    data: metrics
  });

  // Custom metrics endpoint
  await fetch('/api/metrics/chat', {
    method: 'POST',
    body: JSON.stringify(metrics)
  });

  transaction.finish();
}

// Usage in route.ts
const startTime = Date.now();
try {
  const response = await chat.sendMessageStream(/* ... */);
```

```
    await trackChatMetrics({
      sessionId,
      userId,
      latencyMs: Date.now() - startTime,
      cached: false,
      // ...
    });
  } catch (error) {
    await trackChatMetrics({
      sessionId,
      userId,
      latencyMs: Date.now() - startTime,
      error: error.message
    });
  }
```

Dashboard Metrics:

- P50, P95, P99 latency
- AI API error rate
- Cost per user (token usage)
- Cache hit rate
- Learning style distribution

10. Database Query Optimization Audit

Create Composite Indexes:

```
-- For session listing query
CREATE INDEX idx_chat_sessions_user_active_time
  ON chat_sessions(user_id, is_deleted, last_activity DESC)
  INCLUDE (session_token, summary, started_at);

-- For message fetching
CREATE INDEX idx_chat_messages_session_active_time
  ON chat_messages(session_id, is_deleted, created_at ASC)
  INCLUDE (sender, text);

-- For analytics queries
CREATE INDEX idx_chat_sessions_date_range
  ON chat_sessions(started_at)
  WHERE is_deleted = false;
```

Query Performance Testing:

```
-- Before optimization
EXPLAIN ANALYZE
SELECT * FROM chat_sessions
WHERE user_id = 'xxx' AND is_deleted = false
```

```
ORDER BY last_activity DESC;

-- Expected improvement:
-- Before: Seq Scan (cost=0..1000)
-- After: Index Scan using idx_chat_sessions_user_active_time (cost=0..50)
```

11. Implement Circuit Breaker for Webhook

Pattern for Resilience:

```
// lib/resilience/circuit-breaker.ts
enum State { CLOSED, OPEN, HALF_OPEN }

class CircuitBreaker {
  private state = State.CLOSED;
  private failureCount = 0;
  private lastFailureTime = 0;

  constructor(
    private threshold = 5,
    private timeout = 60000,
    private resetTimeout = 30000
  ) {}

  async execute<T>(fn: () => Promise<T>): Promise<T> {
    if (this.state === State.OPEN) {
      if (Date.now() - this.lastFailureTime > this.resetTimeout) {
        this.state = State.HALF_OPEN;
      } else {
        throw new Error('Circuit breaker is OPEN');
      }
    }

    try {
      const result = await fn();
      this.onSuccess();
      return result;
    } catch (error) {
      this.onFailure();
      throw error;
    }
  }

  private onSuccess() {
    this.failureCount = 0;
    this.state = State.CLOSED;
  }

  private onFailure() {
    this.failureCount++;
    this.lastFailureTime = Date.now();
```

```
    if (this.failureCount >= this.threshold) {
      this.state = State.OPEN;
    }
  }
}

// Usage in chatbot.ts
const webhookBreaker = new CircuitBreaker();

try {
  const response = await webhookBreaker.execute(() =>
    fetch(webhookUrl, { /* config */ })
  );
} catch (error) {
  if (error.message === 'Circuit breaker is OPEN') {
    // Fallback to direct AI or cached response
    return res.json({
      error: 'Service temporarily unavailable. Please try again.'
    });
  }
}
```

12. Content Delivery Optimization for Low Bandwidth

Lazy Loading Strategy:

```
// components/chatbot/chatbot-widget.tsx
import dynamic from 'next/dynamic';

const ReactMarkdown = dynamic(() => import('react-markdown'), {
  loading: () => <div className="animate-pulse">Loading...</div>,
  ssr: false
});

const remarkGfm = dynamic(() => import('remark-gfm'));
```

Response Compression:

```
// middleware.ts
import { NextResponse } from 'next/server';
import { compress } from 'zlib';

export function middleware(request: NextRequest) {
  const response = NextResponse.next();

  if (request.headers.get('accept-encoding')?.includes('gzip')) {
    response.headers.set('Content-Encoding', 'gzip');
  }
```

```
  return response;
}
```

Image Attachment Optimization:

```
// Instead of base64
const handleFileUpload = async (files: File[]) => {
  for (const file of files) {
    if (file.type.startsWith('image/')) {
      // Compress before upload
      const compressed = await compressImage(file, {
        maxWidth: 1024,
        quality: 0.8
      });

      // Upload to S3
      const url = await uploadToStorage(compressed);
      attachments.push({ type: 'image', url });
    }
  }
};

// Backend stores URL reference, not base64
```

---

5. Best Practices & Stability Guidelines

Development Workflow

1. **Commit Strategy**
   - ✅ Commit working features incrementally
   - ❌ Don't leave large uncommitted changesets
   - ✅ Use conventional commits: `feat:`, `fix:`, `refactor:`

2. **Testing Requirements**

```
# Before committing
npm run lint
npm run test
npm run build

# Chat-specific tests
npm run test -- chatbot
```

3. **Environment Variable Management**

```
# Repository structure
.env.example          # Template with dummy values
.env.local            # Local dev (gitignored)
.env.production       # Encrypted in CI/CD

# Required in .env.example
GOOGLE_GENERATIVE_AI_API_KEY=your_key_here
NEXT_PUBLIC_API_URL=http://localhost:3001/api
NODE_ENV=development
```

Monitoring Alerts

Set up automated alerts for:

Metric	Threshold	Action
AI API Error Rate	>5%	Page on-call engineer
Average Response Latency	>3s	Investigate performance
Daily Token Usage	>80% of quota	Throttle low-priority requests
Database Connection Pool	>90% utilization	Scale up instances
Chat Session Creation Failures	>1%	Check auth service

Database Maintenance

```sql
-- Weekly vacuum for chat tables
VACUUM ANALYZE chat_sessions;
VACUUM ANALYZE chat_messages;

-- Monthly cleanup of soft-deleted data >90 days old
DELETE FROM chat_sessions
WHERE is_deleted = true
  AND deleted_at < NOW() - INTERVAL '90 days';

-- Index health check
SELECT
  schemaname, tablename, indexname, idx_scan, idx_tup_read
FROM pg_stat_user_indexes
WHERE schemaname = 'public'
  AND tablename LIKE 'chat_%'
ORDER BY idx_scan ASC;
```

Code Quality Standards

AI Integration Rules:

1. **Always** validate AI responses before displaying

2. **Never** trust user input to AI (sanitize first)
3. **Always** implement timeouts for AI calls
4. **Never** expose full error messages to users
5. **Always** log AI interactions for debugging

Example:

```
// ❌ Bad
const response = await ai.generate(userInput);
return response;

// ✅ Good
const sanitized = sanitizeInput(userInput);
const timeoutMs = 15000;

try {
  const response = await Promise.race([
    ai.generate(sanitized),
    timeout(timeoutMs)
  ]);

  const validated = validateAIResponse(response);
  logAIInteraction({ input: sanitized, output: validated });

  return validated;
} catch (error) {
  logError(error);
  return "I'm having trouble responding. Please try again.";
}
```

Security Checklist

- [ ] All API keys in environment variables (not hardcoded)
- [ ] Input validation on all user-generated content
- [ ] Rate limiting on AI endpoints
- [ ] Authentication required for chat endpoints
- [ ] SQL injection prevention (parameterized queries)
- [ ] XSS prevention (markdown sanitization)
- [ ] CORS properly configured
- [ ] Webhook signatures validated (if applicable)
- [ ] File upload size limits enforced
- [ ] Sensitive data encrypted at rest

---

6. Conclusion

Summary of Findings

The recent uncommitted changes represent a **significant enhancement** to the EduLearn chatbot system, introducing:

✅ **Positive Additions:**

- Neuro-adaptive AI for accessibility
- Session management for user experience
- Professional markdown rendering
- Comprehensive chat history

⚠ **Critical Issues Requiring Immediate Attention:**

- Dual AI architecture creating redundancy
- Exposed API keys in version control
- Missing database migration for user learning styles
- No cost control mechanisms for AI API usage

📈 **Performance Concerns:**

- Query optimization needed for scale
- Low-bandwidth users may struggle with bundle size
- No pagination for message loading

Overall Code Quality Rating

Category	Score (1-10)	Notes
Functionality	8/10	Features work but architectural redundancy
Security	4/10	API key exposure is critical issue
Performance	6/10	Works at small scale, will struggle with growth
Maintainability	5/10	Dual implementations increase complexity
Scalability	5/10	Database indexes incomplete, no caching
Low-Bandwidth Readiness	4/10	Large bundles, no compression, no pagination

Overall: 6/10 - Solid feature set with notable technical debt

Priority Action Items

This Week:

1. Remove `.env.local` from git and rotate API key
2. Choose one AI implementation path and deprecate the other
3. Add database migration for `learning_style` field

This Month:

1. Implement rate limiting and cost controls
2. Fix session persistence with localStorage

3. Add composite database indexes
4. Implement message pagination

This Quarter:

1. Add Redis caching layer
2. Implement circuit breaker for webhook
3. Optimize for low-bandwidth with compression
4. Set up comprehensive monitoring

Report Prepared By: Automated System Analysis
Next Review: After implementation of critical fixes
Questions: Contact the development team lead