Contents

SECTION 33: TIME MACHINE UI & SIMPLIFIED AI API (v4.0.0)

Version: 4.0.0 | The visual "fly back through time" experience + AI-friendly API

33.1 Simplified AI API for Time Machine

The AI API allows client apps to let their AI assistants help users navigate history naturally.

```
// packages/functions/src/handlers/thinktank/ai-time-machine.handlers.ts

import { APIGatewayProxyEvent, APIGatewayProxyResult } from 'aws-lambda';
import { TimeMachineService } from '../../services/time-machine.service';
import { pool } from '../../utils/db';

const service = new TimeMachineService(pool);

const corsHeaders = {
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Headers': 'Content-Type,Authorization',
  'Content-Type': 'application/json',
};

//
// AI-FRIENDLY SIMPLIFIED API
// These endpoints return human-readable summaries that AI can relay to users
//

/**
```

```
 * GET /api/ai/chats/:chatId/history/summary
 *
 * Returns a human-readable summary of chat history that an AI can present.
 *
 * Example response:
 * {
 *   "summary": "This conversation has 47 snapshots over 3 days. You've exchanged
 *               156 messages and shared 8 files. The oldest point you can restore
 *               to is December 20th at 2:34 PM.",
 *   "highlights": [
 *     "December 22: Major file update - report_final.xlsx",
 *     "December 21: Long discussion about project requirements",
 *     "December 20: Conversation started"
 *   ],
 *   "canRestore": true,
 *   "oldestDate": "2024-12-20",
 *   "newestDate": "2024-12-23"
 * }
 */
export async function getHistorySummary(event: APIGatewayProxyEvent): Promise<APIGatewayProxyR
  try {
    const chatId = event.pathParameters?.chatId;
    const tenantId = event.requestContext.authorizer?.claims?.['custom:tenant_id'];

    if (!chatId) {
      return { statusCode: 400, headers: corsHeaders, body: JSON.stringify({ error: 'chatId re
    }

    const timeline = await service.getTimeline(chatId, tenantId);

    // Generate human-readable summary
    const dayCount = Object.keys(timeline.snapshotsByDate).length;
    const oldestDate = new Date(timeline.oldestSnapshot).toLocaleDateString('en-US', {
      month: 'long', day: 'numeric', hour: 'numeric', minute: '2-digit'
    });

    // Generate highlights from significant snapshots
    const highlights: string[] = [];
    const fileSnapshots = timeline.snapshots.filter(s =>
      s.trigger === 'file_uploaded' || s.trigger === 'file_generated'
    );
    const restoreSnapshots = timeline.snapshots.filter(s => s.trigger === 'restore_performed')

    // Add recent file activity
    if (fileSnapshots.length > 0) {
      const recent = fileSnapshots[fileSnapshots.length - 1];
      const date = new Date(recent.timestamp).toLocaleDateString('en-US', { month: 'long', day
      highlights.push(`${date}: File activity (${timeline.currentFileCount} files total)`);
```

```
    }

    // Add restore activity
    if (restoreSnapshots.length > 0) {
      highlights.push(`You've restored from history ${restoreSnapshots.length} time(s)`);
    }

    // Add conversation start
    if (timeline.snapshots.length > 0) {
      const first = timeline.snapshots[0];
      const date = new Date(first.timestamp).toLocaleDateString('en-US', { month: 'long', day:
      highlights.push(`${date}: Conversation started`);
    }

    const summary = `This conversation has ${timeline.totalSnapshots} snapshots over ${dayCount
      `You've exchanged ${timeline.currentMessageCount} messages and shared ${timeline.current
      `The oldest point you can restore to is ${oldestDate}.`;

    return {
      statusCode: 200,
      headers: corsHeaders,
      body: JSON.stringify({
        summary,
        highlights,
        canRestore: timeline.totalSnapshots > 1,
        oldestDate: timeline.oldestSnapshot.split('T')[0],
        newestDate: timeline.newestSnapshot.split('T')[0],
        stats: {
          totalSnapshots: timeline.totalSnapshots,
          messageCount: timeline.currentMessageCount,
          fileCount: timeline.currentFileCount,
          totalMediaBytes: timeline.totalMediaBytes,
        },
      }),
    };
  } catch (error: any) {
    console.error('getHistorySummary error:', error);
    return { statusCode: 500, headers: corsHeaders, body: JSON.stringify({ error: error.messag
  }
}

/**
 * POST /api/ai/chats/:chatId/history/find
 *
 * Natural language search through history.
 *
 * Request:
 * { "query": "that spreadsheet from last week" }
```

```
 *
 * Response:
 * {
 *   "found": true,
 *   "description": "I found 'budget_2024.xlsx' from December 18th. Would you like me to resto
 *   "items": [
 *     { "type": "file", "name": "budget_2024.xlsx", "date": "2024-12-18", "id": "..." }
 *   ],
 *   "suggestedActions": ["restore", "download", "show_versions"]
 * }
 */
export async function findInHistory(event: APIGatewayProxyEvent): Promise<APIGatewayProxyResult
  try {
    const chatId = event.pathParameters?.chatId;
    const tenantId = event.requestContext.authorizer?.claims?.['custom:tenant_id'];
    const body = JSON.parse(event.body || '{}');

    if (!chatId || !body.query) {
      return { statusCode: 400, headers: corsHeaders, body: JSON.stringify({ error: 'chatId and
    }

    // Search both messages and files
    const [messages, files] = await Promise.all([
      service.searchMessages(chatId, body.query, tenantId),
      service.searchFiles(chatId, body.query, tenantId),
    ]);

    const items: Array<{
      type: 'message' | 'file';
      name?: string;
      preview?: string;
      date: string;
      id: string;
    }> = [];

    // Add top file results
    for (const file of files.slice(0, 3)) {
      items.push({
        type: 'file',
        name: file.displayName,
        date: file.createdAt.split('T')[0],
        id: file.id,
      });
    }

    // Add top message results
    for (const msg of messages.slice(0, 3)) {
      items.push({
```

```
        type: 'message',
        preview: msg.content.substring(0, 100) + (msg.content.length > 100 ? '...' : ''),
        date: msg.createdAt.split('T')[0],
        id: msg.messageId,
      });
    }

    // Generate description
    let description = '';
    if (items.length === 0) {
      description = `I couldn't find anything matching "${body.query}" in your chat history.`;
    } else if (files.length > 0 && messages.length === 0) {
      description = `I found ${files.length} file${files.length !== 1 ? 's' : ''} matching "${
        `The most recent is '${files[0].displayName}' from ${new Date(files[0].createdAt).toLo
    } else if (messages.length > 0 && files.length === 0) {
      description = `I found ${messages.length} message${messages.length !== 1 ? 's' : ''} men
    } else {
      description = `I found ${files.length} file${files.length !== 1 ? 's' : ''} and ${messag
        `related to "${body.query}".`;
    }

    const suggestedActions: string[] = [];
    if (files.length > 0) {
      suggestedActions.push('download', 'show_versions');
    }
    if (messages.length > 0) {
      suggestedActions.push('jump_to_message');
    }
    if (items.length > 0) {
      suggestedActions.push('restore');
    }

    return {
      statusCode: 200,
      headers: corsHeaders,
      body: JSON.stringify({
        found: items.length > 0,
        description,
        items,
        suggestedActions,
      }),
    };
  } catch (error: any) {
    console.error('findInHistory error:', error);
    return { statusCode: 500, headers: corsHeaders, body: JSON.stringify({ error: error.messag
  }
}
```

```typescript
/**
 * POST /api/ai/chats/:chatId/history/restore
 *
 * AI-assisted restore with natural language confirmation.
 *
 * Request:
 * {
 *   "action": "restore_file",
 *   "fileId": "...",
 *   "confirmed": true
 * }
 *
 * Response:
 * {
 *   "success": true,
 *   "message": "I've restored 'budget_2024.xlsx' to the version from December 18th. Your curr
 *   "undoAvailable": true,
 *   "undoSnapshotId": "..."
 * }
 */
export async function aiRestore(event: APIGatewayProxyEvent): Promise<APIGatewayProxyResult> {
  try {
    const chatId = event.pathParameters?.chatId;
    const userId = event.requestContext.authorizer?.claims?.sub;
    const tenantId = event.requestContext.authorizer?.claims?.['custom:tenant_id'];
    const body = JSON.parse(event.body || '{}');

    if (!chatId || !body.action) {
      return { statusCode: 400, headers: corsHeaders, body: JSON.stringify({ error: 'chatId an
    }

    // Require confirmation for restore actions
    if (!body.confirmed) {
      let confirmMessage = '';

      switch (body.action) {
        case 'restore_file':
          confirmMessage = "I'll restore this file to the selected version. Your current versi
          break;
        case 'restore_message':
          confirmMessage = "I'll restore this message to how it was at the selected point. Con
          break;
        case 'restore_all':
          confirmMessage = "I'll restore the entire conversation to the selected point. All yo
          break;
        default:
          return { statusCode: 400, headers: corsHeaders, body: JSON.stringify({ error: 'Unknow
      }
```

```
  return {
    statusCode: 200,
    headers: corsHeaders,
    body: JSON.stringify({
      needsConfirmation: true,
      message: confirmMessage,
      action: body.action,
    }),
  };
}

// Perform the restore
let result;
let message = '';

switch (body.action) {
  case 'restore_file':
    result = await service.restore({
      chatId,
      targetSnapshotId: body.snapshotId,
      scope: 'single_file',
      fileIds: [body.fileId],
      reason: 'AI-assisted restore',
    }, userId, tenantId);
    message = `I've restored the file. Your previous version has been saved - snapshot ${re
    break;

  case 'restore_message':
    result = await service.restore({
      chatId,
      targetSnapshotId: body.snapshotId,
      scope: 'single_message',
      messageIds: [body.messageId],
      reason: 'AI-assisted restore',
    }, userId, tenantId);
    message = `I've restored the message. Your edit history is preserved.`;
    break;

  case 'restore_all':
    result = await service.restore({
      chatId,
      targetSnapshotId: body.snapshotId,
      scope: 'full_chat',
      reason: 'AI-assisted restore',
    }, userId, tenantId);
    message = `I've restored the conversation to that point. ${result.messagesRestored} mes
      `Don't worry - everything from before is saved and you can restore it anytime.`;
```

```
        break;
    }

    return {
      statusCode: 200,
      headers: corsHeaders,
      body: JSON.stringify({
        success: true,
        message,
        undoAvailable: true,
        undoSnapshotId: result?.previousSnapshotId,
        newVersion: result?.newVersion,
      }),
    };
  } catch (error: any) {
    console.error('aiRestore error:', error);
    return { statusCode: 500, headers: corsHeaders, body: JSON.stringify({ error: error.messag
  }
}

/**
 * GET /api/ai/chats/:chatId/history/compare
 *
 * Compare two points in time - useful for "what changed since..."
 *
 * Query params: from=snapshotId&to=snapshotId (or "current")
 *
 * Response:
 * {
 *   "summary": "Between December 18th and now: 12 new messages, 2 files updated, 1 file added
 *   "changes": {
 *     "messagesAdded": 12,
 *     "messagesEdited": 3,
 *     "messagesDeleted": 1,
 *     "filesAdded": ["report_v2.xlsx"],
 *     "filesUpdated": ["budget.xlsx", "notes.md"],
 *     "filesDeleted": []
 *   }
 * }
 */
export async function compareSnapshots(event: APIGatewayProxyEvent): Promise<APIGatewayProxyRes
  try {
    const chatId = event.pathParameters?.chatId;
    const tenantId = event.requestContext.authorizer?.claims?.['custom:tenant_id'];
    const fromId = event.queryStringParameters?.from;
    const toId = event.queryStringParameters?.to || 'current';

    if (!chatId || !fromId) {
```

```
  return { statusCode: 400, headers: corsHeaders, body: JSON.stringify({ error: 'chatId and
}

// Get states at both points
const fromState = await service.getChatAtSnapshot(chatId, fromId, tenantId);

let toState;
if (toId === 'current') {
  const timeline = await service.getTimeline(chatId, tenantId);
  const currentSnapshotId = timeline.snapshots[timeline.snapshots.length - 1]?.id;
  toState = await service.getChatAtSnapshot(chatId, currentSnapshotId, tenantId);
} else {
  toState = await service.getChatAtSnapshot(chatId, toId, tenantId);
}

// Calculate differences
const fromMessageIds = new Set(fromState.messages.map(m => m.messageId));
const toMessageIds = new Set(toState.messages.map(m => m.messageId));

const messagesAdded = toState.messages.filter(m => !fromMessageIds.has(m.messageId)).lengt
const messagesRemoved = fromState.messages.filter(m => !toMessageIds.has(m.messageId)).leng

const fromFileNames = new Set(fromState.files.map(f => f.originalName));
const toFileNames = new Set(toState.files.map(f => f.originalName));

const filesAdded = toState.files.filter(f => !fromFileNames.has(f.originalName)).map(f => f
const filesRemoved = fromState.files.filter(f => !toFileNames.has(f.originalName)).map(f =>

// Files that exist in both but may have been updated
const filesUpdated = toState.files
  .filter(toFile => {
    const fromFile = fromState.files.find(f => f.originalName === toFile.originalName);
    return fromFile && fromFile.version < toFile.version;
  })
  .map(f => f.originalName);

const fromDate = new Date(fromState.snapshot.timestamp).toLocaleDateString('en-US', {
  month: 'long', day: 'numeric'
});

let summary = `Between ${fromDate} and now: `;
const parts: string[] = [];

if (messagesAdded > 0) parts.push(`${messagesAdded} new message${messagesAdded !== 1 ? 's'
if (filesUpdated.length > 0) parts.push(`${filesUpdated.length} file${filesUpdated.length
if (filesAdded.length > 0) parts.push(`${filesAdded.length} file${filesAdded.length !== 1
if (messagesRemoved > 0) parts.push(`${messagesRemoved} message${messagesRemoved !== 1 ? '
```

```
    summary += parts.length > 0 ? parts.join(', ') + '.' : 'no changes.';

    return {
      statusCode: 200,
      headers: corsHeaders,
      body: JSON.stringify({
        summary,
        changes: {
          messagesAdded,
          messagesRemoved,
          filesAdded,
          filesUpdated,
          filesRemoved,
        },
        fromSnapshot: {
          id: fromState.snapshot.id,
          timestamp: fromState.snapshot.timestamp,
          version: fromState.snapshot.version,
        },
        toSnapshot: {
          id: toState.snapshot.id,
          timestamp: toState.snapshot.timestamp,
          version: toState.snapshot.version,
        },
      }),
    };
  } catch (error: any) {
    console.error('compareSnapshots error:', error);
    return { statusCode: 500, headers: corsHeaders, body: JSON.stringify({ error: error.messag
  }
}
```

33.2 AI API Routes

```
// packages/functions/src/routes/ai-time-machine.routes.ts

import { Router } from './router';
import * as handlers from '../handlers/thinktank/ai-time-machine.handlers';

export function registerAITimeMachineRoutes(router: Router) {
  // Summary and discovery
  router.get('/api/ai/chats/:chatId/history/summary', handlers.getHistorySummary);
  router.post('/api/ai/chats/:chatId/history/find', handlers.findInHistory);

  // AI-assisted restore
  router.post('/api/ai/chats/:chatId/history/restore', handlers.aiRestore);
```

```
  // Comparison
  router.get('/api/ai/chats/:chatId/history/compare', handlers.compareSnapshots);
}
```

33.3 Time Machine Visual UI Components

```
// apps/thinktank/src/components/time-machine/time-machine-overlay.tsx

'use client';

import React, { useState, useEffect, useRef } from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import { X, Calendar, Clock, RotateCcw, Download, Search, ChevronLeft, ChevronRight } from 'lu
import { Button } from '@/components/ui/button';
import { Input } from '@/components/ui/input';
import { useTimeline, useChatAtSnapshot, useRestore } from '@/hooks/use-time-machine';
import { formatDistanceToNow, format, parseISO, isSameDay } from 'date-fns';
import { cn } from '@/lib/utils';

interface TimeMachineOverlayProps {
  chatId: string;
  isOpen: boolean;
  onClose: () => void;
}

/**
 * Time Machine Overlay
 *
 * Apple Time Machine-inspired visual experience:
 * - Messages stack backwards into "space" with perspective
 * - Timeline bar on the right edge
 * - Calendar picker for jumping to dates
 * - Current state at front, past fading into background
 */
export function TimeMachineOverlay({ chatId, isOpen, onClose }: TimeMachineOverlayProps) {
  const { data: timeline, isLoading } = useTimeline(chatId);
  const [selectedSnapshotId, setSelectedSnapshotId] = useState<string | null>(null);
  const [viewMode, setViewMode] = useState<'timeline' | 'calendar'>('timeline');
  const [selectedDate, setSelectedDate] = useState<Date>(new Date());

  const { data: snapshotState } = useChatAtSnapshot(chatId, selectedSnapshotId || undefined);
  const restoreMutation = useRestore();

  // Set initial snapshot to latest
  useEffect(() => {
```

```
  if (timeline?.snapshots.length && !selectedSnapshotId) {
    setSelectedSnapshotId(timeline.snapshots[timeline.snapshots.length - 1].id);
  }
}, [timeline, selectedSnapshotId]);

if (!isOpen) return null;

const currentIndex = timeline?.snapshots.findIndex(s => s.id === selectedSnapshotId) ?? -1;
const selectedSnapshot = timeline?.snapshots[currentIndex];

const handleNavigate = (direction: 'prev' | 'next') => {
  if (!timeline) return;

  const newIndex = direction === 'prev'
    ? Math.max(0, currentIndex - 1)
    : Math.min(timeline.snapshots.length - 1, currentIndex + 1);

  setSelectedSnapshotId(timeline.snapshots[newIndex].id);
};

const handleRestore = async () => {
  if (!selectedSnapshotId || !timeline) return;

  const isLatest = currentIndex === timeline.snapshots.length - 1;
  if (isLatest) return; // Can't restore to current

  await restoreMutation.mutateAsync({
    chatId,
    snapshotId: selectedSnapshotId,
    scope: 'full_chat',
  });

  onClose();
};

return (
  <AnimatePresence>
    <motion.div
      initial={{ opacity: 0 }}
      animate={{ opacity: 1 }}
      exit={{ opacity: 0 }}
      className="fixed inset-0 z-50 bg-black"
    >
      {/* Starfield background (like Time Machine) */}
      <div className="absolute inset-0 overflow-hidden">
        <div className="stars" />
      </div>
```

```
{/* Header */}
<div className="absolute top-0 left-0 right-0 z-10 p-4 flex items-center justify-betwe
  <div className="flex items-center gap-4">
    <Button variant="ghost" size="icon" onClick={onClose} className="text-white">
      <X className="h-6 w-6" />
    </Button>
    <h1 className="text-xl font-semibold text-white">Time Machine</h1>
  </div>

  <div className="flex items-center gap-2">
    <Button
      variant={viewMode === 'timeline' ? 'secondary' : 'ghost'}
      size="sm"
      onClick={() => setViewMode('timeline')}
      className="text-white"
    >
      <Clock className="h-4 w-4 mr-2" />
      Timeline
    </Button>
    <Button
      variant={viewMode === 'calendar' ? 'secondary' : 'ghost'}
      size="sm"
      onClick={() => setViewMode('calendar')}
      className="text-white"
    >
      <Calendar className="h-4 w-4 mr-2" />
      Calendar
    </Button>
  </div>
</div>

{/* Main content area */}
<div className="absolute inset-0 pt-16 pb-24 px-4 flex">
  {/* Message stack with 3D perspective */}
  <div className="flex-1 relative perspective-1000">
    <MessageStack
      messages={snapshotState?.messages || []}
      files={snapshotState?.files || []}
      isLoading={isLoading}
    />
  </div>

  {/* Right sidebar - Timeline or Calendar */}
  <div className="w-64 ml-4">
    {viewMode === 'timeline' ? (
      <TimelineBar
        snapshots={timeline?.snapshots || []}
        selectedId={selectedSnapshotId}
```

```
        onSelect={setSelectedSnapshotId}
      />
    ) : (
      <CalendarPicker
        snapshotsByDate={timeline?.snapshotsByDate || {}}
        selectedDate={selectedDate}
        onSelectDate={(date) => {
          setSelectedDate(date);
          // Find first snapshot on that date
          const dateStr = format(date, 'yyyy-MM-dd');
          const snapshot = timeline?.snapshots.find(s =>
            s.timestamp.startsWith(dateStr)
          );
          if (snapshot) {
            setSelectedSnapshotId(snapshot.id);
          }
        }}
      />
    )}
  </div>
</div>

{/* Bottom control bar */}
<div className="absolute bottom-0 left-0 right-0 p-4 bg-gradient-to-t from-black/80 to-
  <div className="flex items-center justify-between max-w-4xl mx-auto">
    {/* Navigation arrows */}
    <div className="flex items-center gap-2">
      <Button
        variant="outline"
        size="icon"
        onClick={() => handleNavigate('prev')}
        disabled={currentIndex <= 0}
        className="bg-white/10 border-white/20 text-white"
      >
        <ChevronLeft className="h-4 w-4" />
      </Button>
      <Button
        variant="outline"
        size="icon"
        onClick={() => handleNavigate('next')}
        disabled={currentIndex >= (timeline?.snapshots.length || 0) - 1}
        className="bg-white/10 border-white/20 text-white"
      >
        <ChevronRight className="h-4 w-4" />
      </Button>
    </div>

    {/* Snapshot info */}
```

```
            <div className="text-center text-white">
              {selectedSnapshot && (
                <>
                  <div className="text-lg font-medium">
                    {format(parseISO(selectedSnapshot.timestamp), 'MMMM d, yyyy')}
                  </div>
                  <div className="text-sm text-white/60">
                    {format(parseISO(selectedSnapshot.timestamp), 'h:mm a')} ·
                    Version {selectedSnapshot.version} of {timeline?.totalSnapshots}
                  </div>
                </>
              )}
            </div>

            {/* Actions */}
            <div className="flex items-center gap-2">
              <Button
                variant="outline"
                className="bg-white/10 border-white/20 text-white"
                disabled={currentIndex === (timeline?.snapshots.length || 0) - 1}
                onClick={handleRestore}
              >
                <RotateCcw className="h-4 w-4 mr-2" />
                Restore
              </Button>
            </div>
          </div>
        </div>
      </motion.div>
    </AnimatePresence>
  );
}

//
// MESSAGE STACK - 3D perspective view of messages receding into the past
//

function MessageStack({
  messages,
  files,
  isLoading
}: {
  messages: any[];
  files: any[];
  isLoading: boolean;
}) {
  if (isLoading) {
    return (
```

```
    <div className="flex items-center justify-center h-full">
      <div className="text-white/60">Loading history...</div>
    </div>
  );
}

return (
  <div className="relative h-full overflow-hidden">
    {/* 3D perspective container */}
    <div
      className="absolute inset-0 flex flex-col-reverse items-center justify-end pb-8"
      style={{ transformStyle: 'preserve-3d' }}
    >
      {messages.map((message, index) => {
        // Calculate 3D positioning - newer messages at front, older recede
        const depth = index * 0.5; // How far "back" this message is
        const scale = Math.max(0.3, 1 - depth * 0.1);
        const opacity = Math.max(0.2, 1 - depth * 0.15);
        const blur = Math.min(depth * 0.5, 3);
        const yOffset = index * 60; // Vertical stacking
        const zOffset = -depth * 100; // Depth positioning

        return (
          <motion.div
            key={message.id}
            initial={{ opacity: 0, y: 50 }}
            animate={{ opacity, y: yOffset }}
            className={cn(
              "w-full max-w-2xl p-4 rounded-lg mb-2",
              message.role === 'user'
                ? 'bg-blue-600/80 ml-auto mr-0'
                : 'bg-gray-700/80 mr-auto ml-0'
            )}
            style={{
              transform: `scale(${scale}) translateZ(${zOffset}px)`,
              filter: `blur(${blur}px)`,
            }}
          >
            <div className="text-sm text-white/60 mb-1">
              {message.role === 'user' ? 'You' : 'AI'}
            </div>
            <div className="text-white">
              {message.content.length > 200
                ? message.content.substring(0, 200) + '...'
                : message.content}
            </div>
          </motion.div>
        );
```

```
        })}
      </div>

      {/* Files bar at bottom */}
      {files.length > 0 && (
        <div className="absolute bottom-0 left-0 right-0 p-4 bg-gradient-to-t from-black/60">
          <div className="flex gap-2 overflow-x-auto">
            {files.map(file => (
              <div
                key={file.id}
                className="flex-shrink-0 px-3 py-2 bg-white/10 rounded-lg text-white text-sm"
              >
                  {file.displayName}
              </div>
            ))}
          </div>
        </div>
      )}
    </div>
  );
}

//
// TIMELINE BAR - Visual timeline of snapshots
//

function TimelineBar({
  snapshots,
  selectedId,
  onSelect,
}: {
  snapshots: any[];
  selectedId: string | null;
  onSelect: (id: string) => void;
}) {
  const containerRef = useRef<HTMLDivElement>(null);

  // Group snapshots by date
  const groupedByDate: Record<string, any[]> = {};
  for (const snapshot of snapshots) {
    const date = snapshot.timestamp.split('T')[0];
    if (!groupedByDate[date]) {
      groupedByDate[date] = [];
    }
    groupedByDate[date].push(snapshot);
  }

  return (
```

```
    <div ref={containerRef} className="h-full overflow-y-auto pr-2">
      {Object.entries(groupedByDate).reverse().map(([date, dateSnapshots]) => (
        <div key={date} className="mb-4">
          <div className="text-xs text-white/40 mb-2 sticky top-0 bg-black/50 py-1">
            {format(parseISO(date), 'MMM d, yyyy')}
          </div>
          <div className="space-y-1">
            {dateSnapshots.map(snapshot => (
              <button
                key={snapshot.id}
                onClick={() => onSelect(snapshot.id)}
                className={cn(
                  "w-full text-left px-3 py-2 rounded-lg transition-colors",
                  "text-sm text-white/80 hover:bg-white/10",
                  selectedId === snapshot.id && "bg-white/20 ring-1 ring-white/40"
                )}
              >
                <div className="font-medium">
                  {format(parseISO(snapshot.timestamp), 'h:mm a')}
                </div>
                <div className="text-xs text-white/50">
                  {snapshot.messageCount} messages · {snapshot.fileCount} files
                </div>
              </button>
            ))}
          </div>
        </div>
      ))}
    </div>
  );
}

//
// CALENDAR PICKER - Jump to any date
//

function CalendarPicker({
  snapshotsByDate,
  selectedDate,
  onSelectDate,
}: {
  snapshotsByDate: Record<string, number>;
  selectedDate: Date;
  onSelectDate: (date: Date) => void;
}) {
  const [viewMonth, setViewMonth] = useState(selectedDate);

  // Generate calendar grid
```

```
const daysInMonth = new Date(viewMonth.getFullYear(), viewMonth.getMonth() + 1, 0).getDate()
const firstDayOfMonth = new Date(viewMonth.getFullYear(), viewMonth.getMonth(), 1).getDay();

const days: (number | null)[] = [];
for (let i = 0; i < firstDayOfMonth; i++) {
  days.push(null);
}
for (let i = 1; i <= daysInMonth; i++) {
  days.push(i);
}

return (
  <div className="bg-white/5 rounded-lg p-4">
    {/* Month navigation */}
    <div className="flex items-center justify-between mb-4">
      <Button
        variant="ghost"
        size="icon"
        onClick={() => setViewMonth(new Date(viewMonth.getFullYear(), viewMonth.getMonth() -
        className="text-white"
      >
        <ChevronLeft className="h-4 w-4" />
      </Button>
      <div className="text-white font-medium">
        {format(viewMonth, 'MMMM yyyy')}
      </div>
      <Button
        variant="ghost"
        size="icon"
        onClick={() => setViewMonth(new Date(viewMonth.getFullYear(), viewMonth.getMonth() +
        className="text-white"
      >
        <ChevronRight className="h-4 w-4" />
      </Button>
    </div>

    {/* Day labels */}
    <div className="grid grid-cols-7 gap-1 mb-2">
      {['S', 'M', 'T', 'W', 'T', 'F', 'S'].map((day, i) => (
        <div key={i} className="text-center text-xs text-white/40">
          {day}
        </div>
      ))}
    </div>

    {/* Calendar grid */}
    <div className="grid grid-cols-7 gap-1">
      {days.map((day, i) => {
```

```
          if (day === null) {
            return <div key={i} />;
          }

          const date = new Date(viewMonth.getFullYear(), viewMonth.getMonth(), day);
          const dateStr = format(date, 'yyyy-MM-dd');
          const hasSnapshots = snapshotsByDate[dateStr] > 0;
          const isSelected = isSameDay(date, selectedDate);
          const isToday = isSameDay(date, new Date());

          return (
            <button
              key={i}
              onClick={() => hasSnapshots && onSelectDate(date)}
              disabled={!hasSnapshots}
              className={cn(
                "aspect-square flex items-center justify-center rounded-full text-sm",
                "transition-colors",
                hasSnapshots
                  ? "text-white hover:bg-white/20 cursor-pointer"
                  : "text-white/20 cursor-not-allowed",
                isSelected && "bg-blue-600 text-white",
                isToday && !isSelected && "ring-1 ring-white/40"
              )}
            >
              {day}
              {hasSnapshots && !isSelected && (
                <span className="absolute bottom-0.5 w-1 h-1 bg-blue-400 rounded-full" />
              )}
            </button>
          );
        })}
      </div>
    </div>
  );
}
```

33.4 Time Machine Entry Button

```
// apps/thinktank/src/components/time-machine/time-machine-button.tsx

'use client';

import React, { useState } from 'react';
import { Clock } from 'lucide-react';
import { Button } from '@/components/ui/button';
```

```tsx
import { Tooltip, TooltipContent, TooltipTrigger } from '@/components/ui/tooltip';
import { TimeMachineOverlay } from './time-machine-overlay';

interface TimeMachineButtonProps {
  chatId: string;
}

/**
 * Time Machine Entry Button
 *
 * Small, unobtrusive button that opens the full Time Machine experience.
 * Hidden in the chat header, only visible on hover or in Advanced mode.
 */
export function TimeMachineButton({ chatId }: TimeMachineButtonProps) {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <>
      <Tooltip>
        <TooltipTrigger asChild>
          <Button
            variant="ghost"
            size="sm"
            onClick={() => setIsOpen(true)}
            className="h-8 px-2 text-muted-foreground hover:text-foreground"
          >
            <Clock className="h-4 w-4 mr-1.5" />
            <span className="text-xs">Time Machine</span>
          </Button>
        </TooltipTrigger>
        <TooltipContent>
          <p>Browse and restore chat history</p>
        </TooltipContent>
      </Tooltip>

      <TimeMachineOverlay
        chatId={chatId}
        isOpen={isOpen}
        onClose={() => setIsOpen(false)}
      />
    </>
  );
}
```

33.5 React Hooks for Time Machine

```typescript
// apps/thinktank/src/hooks/use-time-machine.ts

import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import { api } from '@/lib/api';
import type { ChatTimeline, RestoreResult, RestoreScope } from '@radiant/shared';

// Get full timeline
export function useTimeline(chatId: string | undefined) {
  return useQuery({
    queryKey: ['time-machine', 'timeline', chatId],
    queryFn: async () => {
      const response = await api.get<ChatTimeline>(`/thinktank/chats/${chatId}/time-machine`);
      return response.data;
    },
    enabled: !!chatId,
    staleTime: 30000, // 30 seconds
  });
}

// Get chat state at specific snapshot
export function useChatAtSnapshot(chatId: string | undefined, snapshotId: string | undefined) {
  return useQuery({
    queryKey: ['time-machine', 'snapshot', chatId, snapshotId],
    queryFn: async () => {
      const response = await api.get(`/thinktank/chats/${chatId}/time-machine/snapshots/${snaps
      return response.data;
    },
    enabled: !!chatId && !!snapshotId,
  });
}

// Get snapshots for a specific date
export function useSnapshotsByDate(chatId: string | undefined, date: string | undefined) {
  return useQuery({
    queryKey: ['time-machine', 'date', chatId, date],
    queryFn: async () => {
      const response = await api.get(`/thinktank/chats/${chatId}/time-machine/calendar/${date}
      return response.data.snapshots;
    },
    enabled: !!chatId && !!date,
  });
}

// Restore mutation
export function useRestore() {
  const queryClient = useQueryClient();
```

```
  return useMutation({
    mutationFn: async ({
      chatId,
      snapshotId,
      scope = 'full_chat',
      messageIds,
      fileIds,
      reason,
    }: {
      chatId: string;
      snapshotId: string;
      scope?: RestoreScope;
      messageIds?: string[];
      fileIds?: string[];
      reason?: string;
    }) => {
      const response = await api.post<RestoreResult>(`/thinktank/chats/${chatId}/time-machine/r
        snapshotId,
        scope,
        messageIds,
        fileIds,
        reason,
      });
      return response.data;
    },
    onSuccess: (_, variables) => {
      // Invalidate all related queries
      queryClient.invalidateQueries({ queryKey: ['time-machine', 'timeline', variables.chatId]
      queryClient.invalidateQueries({ queryKey: ['chat', variables.chatId] });
      queryClient.invalidateQueries({ queryKey: ['messages', variables.chatId] });
    },
  });
}

// Search in history
export function useHistorySearch(chatId: string | undefined, query: string) {
  return useQuery({
    queryKey: ['time-machine', 'search', chatId, query],
    queryFn: async () => {
      const response = await api.get(`/thinktank/chats/${chatId}/time-machine/search`, {
        params: { q: query },
      });
      return response.data;
    },
    enabled: !!chatId && query.length > 2,
  });
}
```

```typescript
// File versions
export function useFileVersions(chatId: string | undefined, fileName: string | undefined) {
  return useQuery({
    queryKey: ['time-machine', 'file-versions', chatId, fileName],
    queryFn: async () => {
      const response = await api.get(`/thinktank/chats/${chatId}/time-machine/files/${encodeURI
      return response.data.versions;
    },
    enabled: !!chatId && !!fileName,
  });
}

// Create export
export function useCreateExport() {
  return useMutation({
    mutationFn: async ({
      chatId,
      format = 'zip',
      includeMedia = true,
      includeVersionHistory = false,
    }: {
      chatId: string;
      format?: 'zip' | 'json' | 'markdown' | 'pdf' | 'html';
      includeMedia?: boolean;
      includeVersionHistory?: boolean;
    }) => {
      const response = await api.post(`/thinktank/chats/${chatId}/time-machine/export`, {
        format,
        includeMedia,
        includeVersionHistory,
      });
      return response.data;
    },
  });
}
```

33.6 CDK Infrastructure Updates

```typescript
// packages/cdk/src/stacks/time-machine-stack.ts

import * as cdk from 'aws-cdk-lib';
import * as s3 from 'aws-cdk-lib/aws-s3';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as apigateway from 'aws-cdk-lib/aws-apigateway';
```

```typescript
import { Construct } from 'constructs';

interface TimeMachineStackProps extends cdk.StackProps {
  environment: string;
  thinktankApi: apigateway.RestApi;
  thinktankLambda: lambda.Function;
}

export class TimeMachineStack extends cdk.Stack {
  public readonly mediaVaultBucket: s3.Bucket;

  constructor(scope: Construct, id: string, props: TimeMachineStackProps) {
    super(scope, id, props);

    //
    // MEDIA VAULT BUCKET - S3 with versioning, never delete
    //

    this.mediaVaultBucket = new s3.Bucket(this, 'MediaVault', {
      bucketName: `radiant-media-vault-${props.environment}-${cdk.Aws.ACCOUNT_ID}`,

      // CRITICAL: Enable versioning for true immutability
      versioned: true,

      // Security
      blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
      encryption: s3.BucketEncryption.S3_MANAGED,

      // CORS for direct uploads
      cors: [{
        allowedHeaders: ['*'],
        allowedMethods: [s3.HttpMethods.PUT, s3.HttpMethods.POST, s3.HttpMethods.GET],
        allowedOrigins: ['*'],
        exposedHeaders: ['ETag', 'x-amz-version-id'],
        maxAge: 3600,
      }],

      // Lifecycle: Move to cheaper storage, NEVER delete
      lifecycleRules: [{
        id: 'TransitionToIntelligentTiering',
        transitions: [{
          storageClass: s3.StorageClass.INTELLIGENT_TIERING,
          transitionAfter: cdk.Duration.days(30),
        }],
        noncurrentVersionTransitions: [{
          storageClass: s3.StorageClass.GLACIER_INSTANT_RETRIEVAL,
          transitionAfter: cdk.Duration.days(90),
        }],
```

```
    // NO expiration - keep forever
  }],

  // Transfer acceleration
  transferAcceleration: true,

  // RETAIN even if stack deleted
  removalPolicy: cdk.RemovalPolicy.RETAIN,
});

// Deny delete actions (belt and suspenders)
this.mediaVaultBucket.addToResourcePolicy(new iam.PolicyStatement({
  sid: 'DenyObjectDeletion',
  effect: iam.Effect.DENY,
  principals: [new iam.AnyPrincipal()],
  actions: ['s3:DeleteObject', 's3:DeleteObjectVersion'],
  resources: [this.mediaVaultBucket.arnForObjects('*')],
  conditions: {
    StringNotEquals: {
      'aws:PrincipalArn': [
        `arn:aws:iam::${cdk.Aws.ACCOUNT_ID}:role/radiant-gdpr-deletion-role`,
      ],
    },
  },
}));

// Grant Lambda read/write (but not delete)
this.mediaVaultBucket.grantReadWrite(props.thinktankLambda);
props.thinktankLambda.addEnvironment('MEDIA_VAULT_BUCKET', this.mediaVaultBucket.bucketName

//
// API ROUTES
//

const api = props.thinktankApi;
const lambdaIntegration = new apigateway.LambdaIntegration(props.thinktankLambda);

// Time Machine base
const chatsResource = api.root.addResource('chats');
const chatResource = chatsResource.addResource('{chatId}');
const timeMachineResource = chatResource.addResource('time-machine');

// Timeline
timeMachineResource.addMethod('GET', lambdaIntegration);

// Snapshots
const snapshotsResource = timeMachineResource.addResource('snapshots');
snapshotsResource.addResource('{snapshotId}').addMethod('GET', lambdaIntegration);
```

```
    // Calendar
    timeMachineResource.addResource('calendar').addResource('{date}').addMethod('GET', lambdaI

    // Restore
    timeMachineResource.addResource('restore').addMethod('POST', lambdaIntegration);

    // Files
    const filesResource = timeMachineResource.addResource('files');
    filesResource.addMethod('GET', lambdaIntegration);
    filesResource.addResource('{fileName}').addResource('versions').addMethod('GET', lambdaInte

    // Search
    timeMachineResource.addResource('search').addMethod('GET', lambdaIntegration);

    // Export
    timeMachineResource.addResource('export').addMethod('POST', lambdaIntegration);

    // File download
    api.root.addResource('files').addResource('{fileId}').addResource('download').addMethod('G

    // Export status
    api.root.addResource('exports').addResource('{bundleId}').addMethod('GET', lambdaIntegrati

    //
    // AI API ROUTES
    //

    const aiResource = api.root.addResource('ai');
    const aiChatsResource = aiResource.addResource('chats').addResource('{chatId}');
    const historyResource = aiChatsResource.addResource('history');

    historyResource.addResource('summary').addMethod('GET', lambdaIntegration);
    historyResource.addResource('find').addMethod('POST', lambdaIntegration);
    historyResource.addResource('restore').addMethod('POST', lambdaIntegration);
    historyResource.addResource('compare').addMethod('GET', lambdaIntegration);

    //
    // OUTPUTS
    //

    new cdk.CfnOutput(this, 'MediaVaultBucketName', {
      value: this.mediaVaultBucket.bucketName,
      description: 'Time Machine Media Vault S3 Bucket',
    });
  }
}
```

33.7 Integration with Think Tank Chat

```
// apps/thinktank/src/components/chat/chat-header.tsx (updated)

import { TimeMachineButton } from '../time-machine/time-machine-button';

export function ChatHeader({ chatId, title }: ChatHeaderProps) {
  return (
    <div className="flex items-center justify-between p-4 border-b">
      <div className="flex items-center gap-2">
        <h1 className="font-semibold">{title || 'New Chat'}</h1>
      </div>
      <div className="flex items-center gap-2">
        {/* Time Machine - hidden until hover */}
        <div className="opacity-0 group-hover:opacity-100 transition-opacity">
          <TimeMachineButton chatId={chatId} />
        </div>

        {/* Other header actions */}
        <Button variant="ghost" size="sm">
          <MoreHorizontal className="h-4 w-4" />
        </Button>
      </div>
    </div>
  );
}
```