Backend Code Documentation

Complete Backend & Logic Documentation (All 4 Phases)

Generated on: October 31, 2025 at 17:08:11

Phase 1: Core Backend Schema, Models & Configuration

This phase documents the core database schema, type definitions, and configuration files that form the foundation of the backend system.

■ File: convex/schema.ts

Size: 7535 bytes | Last Modified: 2025-10-31 16:47:59

■ ***File Overview:***
■ ***Database Schema Definition*** *- Defines data models and table structures*
■ *Contains: 2 imports, 1 exports, ~0 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 1
import { defineSchema, defineTable } from "convex/server";
import { v } from "convex/values";

export default defineSchema({
    users: defineTable({
        userId: v.string(), // clerkId
        email: v.string(),
        name: v.string(),
        isPro: v.boolean(),
        proSince: v.optional(v.number()),
        lemonSqueezyCustomerId: v.optional(v.string()),
        lemonSqueezyOrderId: v.optional(v.string()),
    }).index("by_user_id", ["userId"]),

    codeExecutions: defineTable({
        userId: v.string(),
        language: v.string(),
        code: v.string(),
        output: v.optional(v.string()),
        error: v.optional(v.string()),
    }).index("by_user_id", ["userId"]),

    snippets: defineTable({
        userId: v.string(),
        title: v.string(),
        language: v.string(),
        code: v.string(),
        userName: v.string(), // store user's name for easy access
    }).index("by_user_id", ["userId"]),

    snippetComments: defineTable({
        snippetId: v.id("snippets"),
        userId: v.string(),
        userName: v.string(),
        content: v.string(), // This will store HTML content
    }).index("by_snippet_id", ["snippetId"]),

    stars: defineTable({
        userId: v.string(),
        snippetId: v.id("snippets"),
    })
        .index("by_user_id", ["userId"])
        .index("by_snippet_id", ["snippetId"])
        .index("by_user_id_and_snippet_id", ["userId", "snippetId"]),
```

```
// File System Tables
folders: defineTable({
    userId: v.string(),
    name: v.string(),
    parentFolderId: v.optional(v.id("folders")),
    isShared: v.boolean(),
    createdAt: v.number(),
    path: v.string(), // full path like "/DSA Practice/Arrays"
})
    .index("by_user_id", ["userId"])
    .index("by_parent_folder", ["parentFolderId"]),

practiceFiles: defineTable({
    userId: v.string(),
    folderId: v.optional(v.id("folders")),
    name: v.string(),
    language: v.string(),
    code: v.string(),
    description: v.optional(v.string()),
    isShared: v.boolean(),
    createdAt: v.number(),
    updatedAt: v.number(),
    path: v.string(), // full file path
})
    .index("by_user_id", ["userId"])
    .index("by_folder_id", ["folderId"])
    .index("by_user_and_folder", ["userId", "folderId"]),

// Collaboration Tables
collaborativeSessions: defineTable({
    name: v.string(),
    creatorId: v.string(),
    sessionKey: v.string(), // URL-friendly unique identifier
    language: v.string(),
    code: v.string(),
    description: v.optional(v.string()), // Session description
    isPublic: v.boolean(),
    isActive: v.boolean(),
    activeUsers: v.array(v.string()),
    maxUsers: v.number(),
    createdAt: v.number(),
    lastActivity: v.number(),
    status: v.optional(v.union(v.literal("active"), v.literal("inactive"), v.literal("scheduled_for_dele
    expiresAt: v.optional(v.number()),
    originalSavedSessionId: v.optional(v.id("savedSessions")), // Reference to saved session if loaded f
    // Additional session settings
    sessionSettings: v.optional(v.object({
        allowGuests: v.optional(v.boolean()),
        autoSave: v.optional(v.boolean()),
        theme: v.optional(v.string()),
    })),
})
    .index("by_creator_id", ["creatorId"])
    .index("by_is_public", ["isPublic"])
    .index("by_is_active", ["isActive"])
    .index("by_status", ["status"])
    .index("by_expires_at", ["expiresAt"])
    .index("by_session_key", ["sessionKey"])
    .index("by_original_saved_session", ["originalSavedSessionId"]),

sessionParticipants: defineTable({
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
    userName: v.string(),
    permission: v.union(v.literal("read"), v.literal("write")),
    joinedAt: v.number(),
    lastActive: v.number(),
    isActive: v.boolean(),
    lastSeen: v.optional(v.number()), // Make this optional too for existing records
})
    .index("by_session_id", ["sessionId"])
    .index("by_user_id", ["userId"])
    .index("by_session_and_user", ["sessionId", "userId"])
    .index("by_last_active", ["lastActive"])
    .index("by_is_active", ["isActive"]),

// Real-time code operations for collaboration
codeOperations: defineTable({
    sessionId: v.id("collaborativeSessions"),
```

```
        userId: v.string(),
        operation: v.object({
            type: v.string(), // "insert", "delete", "replace"
            position: v.number(),
            content: v.string(),
            length: v.optional(v.number()),
        }),
        timestamp: v.number(),
    }).index("by_session_id", ["sessionId"]),

    // Session chat messages
    sessionMessages: defineTable({
        sessionId: v.id("collaborativeSessions"),
        userId: v.string(),
        userName: v.string(),
        message: v.string(),
        timestamp: v.number(),
    }).index("by_session_id", ["sessionId"]),

    // Saved sessions - persistent user sessions
    savedSessions: defineTable({
        userId: v.string(),
        originalSessionId: v.optional(v.id("collaborativeSessions")), // Track original session
        name: v.string(),
        language: v.string(),
        code: v.string(),
        description: v.optional(v.string()),
        createdAt: v.number(),
        updatedAt: v.number(),
        tags: v.optional(v.array(v.string())),
        isPrivate: v.boolean(),
        // Session metadata to preserve when loading
        maxUsers: v.optional(v.number()), // Maximum users allowed in the session
        sessionSettings: v.optional(v.object({
            allowGuests: v.optional(v.boolean()),
            autoSave: v.optional(v.boolean()),
            theme: v.optional(v.string()),
        })),
    }).index("by_user_id", ["userId"])
      .index("by_created_at", ["createdAt"])
      .index("by_updated_at", ["updatedAt"])
      .index("by_user_and_original_session", ["userId", "originalSessionId"]),

    // Session-specific file system for collaborative sessions
    sessionFolders: defineTable({
        sessionId: v.id("collaborativeSessions"),
        userId: v.string(), // Creator of the folder
        name: v.string(),
        parentFolderId: v.optional(v.id("sessionFolders")),
        createdAt: v.number(),
        path: v.string(), // full path like "/utils/helpers"
    })
        .index("by_session_id", ["sessionId"])
        .index("by_parent_folder", ["parentFolderId"])
        .index("by_session_and_parent", ["sessionId", "parentFolderId"]),

    sessionFiles: defineTable({
        sessionId: v.id("collaborativeSessions"),
        userId: v.string(), // Creator of the file
        folderId: v.optional(v.id("sessionFolders")),
        name: v.string(),
        language: v.string(),
        code: v.string(),
        description: v.optional(v.string()),
        createdAt: v.number(),
        updatedAt: v.number(),
        path: v.string(), // full file path
    })
        .index("by_session_id", ["sessionId"])
        .index("by_folder_id", ["folderId"])
        .index("by_session_and_folder", ["sessionId", "folderId"]),
});
```

■ File: convex/_generated/dataModel.d.ts

Size: 1820 bytes | Last Modified: 2025-10-31 16:47:59

■ ***File Overview:***
■ *Contains: 3 imports, 4 exports, ~0 functions/constants*
■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 1
/* eslint-disable */
/**
 * Generated data model types.
 *
 * THIS CODE IS AUTOMATICALLY GENERATED.
 *
 * To regenerate, run `npx convex dev`.
 * @module
 */

import type {
  DataModelFromSchemaDefinition,
  DocumentByName,
  TableNamesInDataModel,
  SystemTableNames,
} from "convex/server";
import type { GenericId } from "convex/values";
import schema from "../schema.js";

/**
 * The names of all of your Convex tables.
 */
export type TableNames = TableNamesInDataModel<DataModel>;

/**
 * The type of a document stored in Convex.
 *
 * @typeParam TableName - A string literal type of the table name (like "users").
 */
export type Doc<TableName extends TableNames> = DocumentByName<
  DataModel,
  TableName
>;

/**
 * An identifier for a document in Convex.
 *
 * Convex documents are uniquely identified by their `Id`, which is accessible
 * on the `_id` field. To learn more, see [Document IDs](https://docs.convex.dev/using/document-ids).
 *
 * Documents can be loaded using `db.get(id)` in query and mutation functions.
 *
 * IDs are just strings at runtime, but this type can be used to distinguish them from other
 * strings when type checking.
 *
 * @typeParam TableName - A string literal type of the table name (like "users").
 */
export type Id<TableName extends TableNames | SystemTableNames> =
  GenericId<TableName>;

/**
 * A type describing your Convex data model.
 *
 * This type includes information about what tables you have, the type of
 * documents stored in those tables, and the indexes defined on them.
 *
 * This type is used to parameterize methods like `queryGeneric` and
 * `mutationGeneric` to make them type-safe.
 */
export type DataModel = DataModelFromSchemaDefinition<typeof schema>;
```

■ File: src/types/index.ts

Size: 1458 bytes | Last Modified: 2025-10-31 16:47:59

■ ***File Overview:***
■ *Contains: 2 imports, 7 exports, ~0 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Monaco code editor*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 1
import { Monaco } from "@monaco-editor/react";
import { Id } from "../../convex/_generated/dataModel";

export interface Theme {
    id: string;
    label: string;
    color: string;
}

export interface Language {
    id: string;
    label: string;
    logoPath: string;
    monacoLanguage: string;
    defaultCode: string;
    pistonRuntime: LanguageRuntime;
}

export interface LanguageRuntime {
    language: string;
    version: string;
}

export interface ExecuteCodeResponse {
    compile?: {
        output: string;
    };
    run?: {
        output: string;
        stderr: string;
    };
}

export interface ExecutionResult {
    code: string;
    output: string;
    error: string | null;
}

export interface CodeEditorState {
    language: string;
    output: string;
    isRunning: boolean;
    error: string | null;
    theme: string;
    fontSize: number;
    editor: Monaco | null;
    executionResult: ExecutionResult | null;

    setEditor: (editor: Monaco) => void;
    getCode: () => string;
    setLanguage: (language: string) => void;
    setTheme: (theme: string) => void;
    setFontSize: (fontSize: number) => void;
    runCode: () => Promise<void>;
}

export interface Snippet {
    _id: Id<"snippets">;
    _creationTime: number;
```

```
    userId: string;
    language: string;
    code: string;
    title: string;
    userName: string;
}
```

■ File: convex/auth.config.ts

Size: 186 bytes | Last Modified: 2025-10-31 16:47:59

> ■ ***File Overview:***
> ■ *Contains: 0 imports, 1 exports, ~0 functions/constants*
> ■ *Uses Convex backend framework*
> ■ *Integrates Clerk authentication*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 1
export default {
  providers: [
    {
      domain: "https://decent-lemur-48.clerk.accounts.dev/",
      applicationID: "convex",
    },
  ],
};
```

■ File: next.config.ts

Size: 175 bytes | Last Modified: 2025-10-31 16:47:59

> ■ ***File Overview:***
> ■ *Contains: 1 imports, 1 exports, ~0 functions/constants*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 1
import type { NextConfig } from "next";

const nextConfig: NextConfig = {
  /* config options here */
};

export default nextConfig;
```

■ File: vercel.json

Size: 804 bytes | Last Modified: 2025-10-23 13:49:57

■ ***File Overview:***
■ *Contains: 0 imports, 0 exports, ~0 functions/constants*

■ Complete Code:

```
{
  "framework": "nextjs",
  "rewrites": [
    {
      "source": "/(.*)",
      "destination": "/"
    }
  ],
  "headers": [
    {
      "source": "/api/(.*)",
      "headers": [
        {
          "key": "Access-Control-Allow-Credentials",
          "value": "true"
        },
        {
          "key": "Access-Control-Allow-Origin",
          "value": "*"
        },
        {
          "key": "Access-Control-Allow-Methods",
          "value": "GET,OPTIONS,PATCH,DELETE,POST,PUT"
        },
        {
          "key": "Access-Control-Allow-Headers",
          "value": "X-CSRF-Token, X-Requested-With, Accept, Accept-Version, Content-Length, Content-MD5, Co
        }
      ]
    }
  ],
  "cleanUrls": true,
  "trailingSlash": false
}
```

■ File: tsconfig.json

Size: 629 bytes | Last Modified: 2025-10-23 13:49:57

■ ***File Overview:***
■ *Contains: 0 imports, 0 exports, ~0 functions/constants*

■ Complete Code:

```
{
  "compilerOptions": {
    "target": "ES2017",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "strict": true,
    "noEmit": true,
    "esModuleInterop": true,
    "module": "esnext",
    "moduleResolution": "bundler",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "jsx": "preserve",
    "incremental": true,
    "plugins": [
```

```
      {
        "name": "next"
      }
    ],
    "paths": {
      "@/*": ["./src/*"]
    }
  },
  "include": ["next-env.d.ts", "**/*.ts", "**/*.tsx", ".next/types/**/*.ts"],
  "exclude": ["node_modules"]
}
```

Phase 2: Backend Business Logic & Core Operations

This phase documents all backend business logic including mutations, queries, action functions, code execution, collaboration features, file/folder operations, user management, payment integration, and scheduled jobs.

■ File: convex/codeExecution.ts

Size: 3845 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■ *Contains: 3 imports, 3 exports, ~3 functions/constants*
> ■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { ConvexError, v } from "convex/values";
import { mutation, query } from "./_generated/server";
import { paginationOptsValidator } from "convex/server";

export const saveExecution = mutation({
    args: {
        language: v.string(),
        code: v.string(),
        // we could have either one of them, or both at the same time
        output: v.optional(v.string()),
        error: v.optional(v.string()),
    },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) throw new ConvexError("Not authenticated");

        // check pro status
        const user = await ctx.db
            .query("users")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), identity.subject))
            .first();

        if (!user?.isPro && args.language !== "javascript") {
            throw new ConvexError("Pro subscription required to use this language");
        }

        await ctx.db.insert("codeExecutions", {
            ...args,
            userId: identity.subject,
        });
    },
});

export const getUserExecutions = query({
    args: {
        userId: v.string(),
        paginationOpts: paginationOptsValidator,
    },
    handler: async (ctx, args) => {
        return await ctx.db
            .query("codeExecutions")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), args.userId))
            .order("desc")
            .paginate(args.paginationOpts);
    },
});
```

```
export const getUserStats = query({
    args: { userId: v.string() },
    handler: async (ctx, args) => {
        const executions = await ctx.db
            .query("codeExecutions")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), args.userId))
            .collect();

        // Get starred snippets
        const starredSnippets = await ctx.db
            .query("stars")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), args.userId))
            .collect();

        // Get all starred snippet details to analyze languages
        const snippetIds = starredSnippets.map((star) => star.snippetId);
        const snippetDetails = await Promise.all(snippetIds.map((id) => ctx.db.get(id)));

        // Calculate most starred language
        const starredLanguages = snippetDetails.filter(Boolean).reduce(
            (acc, curr) => {
                if (curr?.language) {
                    acc[curr.language] = (acc[curr.language] || 0) + 1;
                }
                return acc;
            },
            {} as Record<string, number>
        );

        const mostStarredLanguage =
            Object.entries(starredLanguages).sort(([, a], [, b]) => b - a)[0]?.[0] ?? "N/A";

        // Calculate execution stats
        const last24Hours = executions.filter(
            (e) => e._creationTime > Date.now() - 24 * 60 * 60 * 1000
        ).length;

        const languageStats = executions.reduce(
            (acc, curr) => {
                acc[curr.language] = (acc[curr.language] || 0) + 1;
                return acc;
            },
            {} as Record<string, number>
        );

        const languages = Object.keys(languageStats);
        const favoriteLanguage = languages.length
            ? languages.reduce((a, b) => (languageStats[a] > languageStats[b] ? a : b))
            : "N/A";

        return {
            totalExecutions: executions.length,
            languagesCount: languages.length,
            languages: languages,
            last24Hours,
            favoriteLanguage,
            languageStats,
            mostStarredLanguage,
        };
    },
});
```

■ File: convex/codeExecutions.ts

Size: 3845 bytes | Last Modified: 2025-10-31 16:49:43

■ ***File Overview:***
■ *Contains: 3 imports, 3 exports, ~3 functions/constants*
■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { ConvexError, v } from "convex/values";
import { mutation, query } from "./_generated/server";
import { paginationOptsValidator } from "convex/server";

export const saveExecution = mutation({
    args: {
        language: v.string(),
        code: v.string(),
        // we could have either one of them, or both at the same time
        output: v.optional(v.string()),
        error: v.optional(v.string()),
    },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) throw new ConvexError("Not authenticated");

        // check pro status
        const user = await ctx.db
            .query("users")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), identity.subject))
            .first();

        if (!user?.isPro && args.language !== "javascript") {
            throw new ConvexError("Pro subscription required to use this language");
        }

        await ctx.db.insert("codeExecutions", {
            ...args,
            userId: identity.subject,
        });
    },
});

export const getUserExecutions = query({
    args: {
        userId: v.string(),
        paginationOpts: paginationOptsValidator,
    },
    handler: async (ctx, args) => {
        return await ctx.db
            .query("codeExecutions")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), args.userId))
            .order("desc")
            .paginate(args.paginationOpts);
    },
});

export const getUserStats = query({
    args: { userId: v.string() },
    handler: async (ctx, args) => {
        const executions = await ctx.db
            .query("codeExecutions")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), args.userId))
            .collect();

        // Get starred snippets
        const starredSnippets = await ctx.db
            .query("stars")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), args.userId))
            .collect();

        // Get all starred snippet details to analyze languages
        const snippetIds = starredSnippets.map((star) => star.snippetId);
        const snippetDetails = await Promise.all(snippetIds.map((id) => ctx.db.get(id)));

        // Calculate most starred language
        const starredLanguages = snippetDetails.filter(Boolean).reduce(
            (acc, curr) => {
                if (curr?.language) {
                    acc[curr.language] = (acc[curr.language] || 0) + 1;
```

```
                }
                return acc;
            },
            {} as Record<string, number>
        );

        const mostStarredLanguage =
            Object.entries(starredLanguages).sort(([, a], [, b]) => b - a)[0]?.[0] ?? "N/A";

        // Calculate execution stats
        const last24Hours = executions.filter(
            (e) => e._creationTime > Date.now() - 24 * 60 * 60 * 1000
        ).length;

        const languageStats = executions.reduce(
            (acc, curr) => {
                acc[curr.language] = (acc[curr.language] || 0) + 1;
                return acc;
            },
            {} as Record<string, number>
        );

        const languages = Object.keys(languageStats);
        const favoriteLanguage = languages.length
            ? languages.reduce((a, b) => (languageStats[a] > languageStats[b] ? a : b))
            : "N/A";

        return {
            totalExecutions: executions.length,
            languagesCount: languages.length,
            languages: languages,
            last24Hours,
            favoriteLanguage,
            languageStats,
            mostStarredLanguage,
        };
    },
});
```

■ File: convex/collaboration.ts

Size: 36768 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■■ ***Business Logic*** *- Core application functionality*
> ■ *Contains: 2 imports, 30 exports, ~6 functions/constants*
> ■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { mutation, query } from "./_generated/server";
import { v } from "convex/values";

// Generate a unique session key for URL routing
function generateSessionKey(): string {
  const timestamp = Date.now().toString(36);
  const random1 = Math.random().toString(36).substring(2, 8);
  const random2 = Math.random().toString(36).substring(2, 6);
  return `${timestamp}${random1}${random2}`.toLowerCase();
}

// Create a collaborative session
export const createSession = mutation({
  args: {
    name: v.string(),
    creatorId: v.string(),
    language: v.string(),
```

```
    code: v.optional(v.string()),
    description: v.optional(v.string()),
    isPublic: v.boolean(),
    maxUsers: v.optional(v.number()),
    sessionSettings: v.optional(v.object({
      allowGuests: v.optional(v.boolean()),
      autoSave: v.optional(v.boolean()),
      theme: v.optional(v.string()),
    })),
  },
  handler: async (ctx, args) => {
    const {
      name,
      creatorId,
      language,
      code,
      description,
      isPublic,
      maxUsers = 5,
      sessionSettings
    } = args;

    // Check session limit before creating (max 2 total sessions per user)
    const totalSessions = await ctx.db
      .query("collaborativeSessions")
      .withIndex("by_creator_id", (q) => q.eq("creatorId", creatorId))
      .collect();

    if (totalSessions.length >= 2) {
      throw new Error("You can only have 2 sessions maximum. Please delete an existing session first.");
    }

    const now = Date.now();
    const sessionKey = generateSessionKey();

    const sessionId = await ctx.db.insert("collaborativeSessions", {
      name,
      creatorId,
      sessionKey,
      language,
      code: code || getDefaultCodeForLanguage(language),
      description,
      isPublic,
      isActive: true,
      activeUsers: [creatorId],
      maxUsers,
      createdAt: now,
      lastActivity: now,
      status: "active",
      sessionSettings: sessionSettings || {
        allowGuests: false,
        autoSave: true,
        theme: "dark",
      },
    });

    // Add creator as participant
    await ctx.db.insert("sessionParticipants", {
      sessionId,
      userId: creatorId,
      userName: await getUserName(ctx, creatorId),
      permission: "write",
      joinedAt: now,
      lastActive: now,
      lastSeen: now,
      isActive: true,
    });

    return { sessionId, sessionKey };
  },
});

// Join a collaborative session
export const joinSession = mutation({
  args: {
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
```

```
    const { sessionId, userId } = args;

    const session = await ctx.db.get(sessionId);
    if (!session) {
      throw new Error("Session not found");
    }

    // Allow joining inactive sessions - they will be reactivated
    if (session.status === "scheduled_for_deletion" || !session.isActive) {
      // Reactivate the session
      await ctx.db.patch(sessionId, {
        status: "active",
        isActive: true,
        expiresAt: undefined,
        lastActivity: Date.now(),
      });
    }

    if (session.activeUsers.length >= session.maxUsers && !session.activeUsers.includes(userId)) {
      throw new Error("Session is full");
    }

    // Check if user is already in session
    const existingParticipant = await ctx.db
      .query("sessionParticipants")
      .withIndex("by_session_and_user", (q) =>
        q.eq("sessionId", sessionId).eq("userId", userId)
      )
      .first();

    if (existingParticipant) {
      // Reactivate existing participant
      await ctx.db.patch(existingParticipant._id, {
        isActive: true,
        lastActive: Date.now(),
        lastSeen: Date.now(),
      });
    } else {
      // Add new participant
      await ctx.db.insert("sessionParticipants", {
        sessionId,
        userId,
        userName: await getUserName(ctx, userId),
        permission: "write", // Default permission
        joinedAt: Date.now(),
        lastActive: Date.now(),
        lastSeen: Date.now(),
        isActive: true,
      });
    }

    // Update session active users
    const updatedActiveUsers = [...new Set([...session.activeUsers, userId])];
    await ctx.db.patch(sessionId, {
      activeUsers: updatedActiveUsers,
      lastActivity: Date.now(),
    });

    return { success: true };
  },
});

// Leave a collaborative session
export const leaveSession = mutation({
  args: {
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { sessionId, userId } = args;

    const session = await ctx.db.get(sessionId);
    if (!session) {
      throw new Error("Session not found");
    }

    // Find and deactivate participant
    const participant = await ctx.db
      .query("sessionParticipants")
```

```
      .withIndex("by_session_and_user", (q) =>
        q.eq("sessionId", sessionId).eq("userId", userId)
      )
      .first();

    if (participant) {
      await ctx.db.patch(participant._id, {
        isActive: false,
        lastActive: Date.now(),
        lastSeen: Date.now(),
      });
    }

    // Update session active users
    const updatedActiveUsers = session.activeUsers.filter(id => id !== userId);
    await ctx.db.patch(sessionId, {
      activeUsers: updatedActiveUsers,
      lastActivity: Date.now(),
    });

    // Check if any users are still active and update session status accordingly
    const now = Date.now();
    const INACTIVE_THRESHOLD = 5 * 60 * 1000; // 5 minutes

    const allParticipants = await ctx.db
      .query("sessionParticipants")
      .withIndex("by_session_id", (q) => q.eq("sessionId", sessionId))
      .collect();

    const activeParticipants = allParticipants.filter(
      p => p.isActive && (p.lastSeen ? now - p.lastSeen <= INACTIVE_THRESHOLD : now - p.lastActive <= INACTI
    );

    // If no active users left, schedule session for deletion
    if (activeParticipants.length === 0) {
      await ctx.db.patch(sessionId, {
        status: "scheduled_for_deletion",
        isActive: false,
        expiresAt: now + (60 * 60 * 1000), // 1 hour from now
      });
    }

    return { success: true };
  },
});

// Update session code
export const updateSessionCode = mutation({
  args: {
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
    code: v.string(),
    operation: v.optional(v.object({
      type: v.string(),
      position: v.number(),
      content: v.string(),
      length: v.optional(v.number()),
    })),
  },
  handler: async (ctx, args) => {
    const { sessionId, userId, code, operation } = args;

    const session = await ctx.db.get(sessionId);
    if (!session) {
      throw new Error("Session not found");
    }

    // Check if user has write permission
    const participant = await ctx.db
      .query("sessionParticipants")
      .withIndex("by_session_and_user", (q) =>
        q.eq("sessionId", sessionId).eq("userId", userId)
      )
      .first();

    if (!participant || participant.permission !== "write") {
      throw new Error("Permission denied");
    }
```

```
    // Only update if the code has actually changed to prevent unnecessary updates
    if (session.code !== code) {
      // Update session code
      await ctx.db.patch(sessionId, {
        code,
        lastActivity: Date.now(),
      });

      // Log the operation for potential conflict resolution
      if (operation) {
        await ctx.db.insert("codeOperations", {
          sessionId,
          userId,
          operation,
          timestamp: Date.now(),
        });
      }
    }

    // Update participant last active (but less frequently to reduce writes)
    const now = Date.now();
    if (now - participant.lastActive > 5000) { // Only update every 5 seconds
      await ctx.db.patch(participant._id, {
        lastActive: now,
      });
    }

    return { success: true };
  },
});

// Get session by sessionKey (for UR

... (truncated 27581 characters)
```

■ File: convex/sessionActivity.ts

Size: 10240 bytes | Last Modified: 2025-10-31 16:49:43

■ ***File Overview:***
■■ ***Business Logic*** *- Core application functionality*
■ *Contains: 2 imports, 6 exports, ~3 functions/constants*
■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { v } from "convex/values";
import { mutation, query, internalMutation } from "./_generated/server";

// Constants for session management
const INACTIVE_THRESHOLD = 5 * 60 * 1000; // 5 minutes - consider user inactive
const EXPIRY_DELAY = 60 * 60 * 1000; // 1 hour - time to wait before deleting inactive session

// Update user's activity in a session (heartbeat)
export const updateUserActivity = mutation({
  args: {
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const now = Date.now();

    // Find the participant
    const participant = await ctx.db
      .query("sessionParticipants")
      .withIndex("by_session_and_user", (q) =>
        q.eq("sessionId", args.sessionId).eq("userId", args.userId)
```

```
      )
      .first();

    if (participant) {
      // Update participant's activity
      await ctx.db.patch(participant._id, {
        lastSeen: now,
        lastActive: now,
        isActive: true,
      });

      // Update session's last activity
      await ctx.db.patch(args.sessionId, {
        lastActivity: now,
      });

      // If session was scheduled for deletion, reactivate it
      const session = await ctx.db.get(args.sessionId);
      if (session && (session.status === "scheduled_for_deletion" || session.status === "inactive")) {
        await ctx.db.patch(args.sessionId, {
          status: "active",
          isActive: true,
          expiresAt: undefined,
        });
      } else if (session && !session.status) {
        // Add status field to existing sessions without it
        await ctx.db.patch(args.sessionId, {
          status: "active",
        });
      }
    }

    return { success: true };
  },
});

// Check for inactive users and manage session status
export const checkSessionActivity = mutation({
  args: {
    sessionId: v.id("collaborativeSessions"),
  },
  handler: async (ctx, args) => {
    const now = Date.now();

    // Get all participants for this session
    const participants = await ctx.db
      .query("sessionParticipants")
      .withIndex("by_session_id", (q) => q.eq("sessionId", args.sessionId))
      .collect();

    // Count active participants
    let activeParticipants = 0;
    const activeUserIds: string[] = [];

    for (const participant of participants) {
      const isActiveNow = participant.lastSeen
        ? now - participant.lastSeen <= INACTIVE_THRESHOLD
        : now - participant.lastActive <= INACTIVE_THRESHOLD;

      if (participant.isActive !== isActiveNow) {
        // Update participant status if it changed
        await ctx.db.patch(participant._id, {
          isActive: isActiveNow,
          lastActive: isActiveNow ? participant.lastActive : now,
        });
      }

      if (isActiveNow) {
        activeParticipants++;
        activeUserIds.push(participant.userId);
      }
    }

    // Get current session
    const session = await ctx.db.get(args.sessionId);
    if (!session) return { success: false, error: "Session not found" };

    // Update session based on active participants
    if (activeParticipants === 0) {
```

```
      // No active users - schedule session for deletion
      if (!session.status || session.status !== "scheduled_for_deletion") {
        await ctx.db.patch(args.sessionId, {
          status: "scheduled_for_deletion",
          isActive: false,
          activeUsers: [],
          expiresAt: now + EXPIRY_DELAY,
          lastActivity: now,
        });
      }
    } else {
      // Has active users - ensure session is active
      if (!session.status || session.status !== "active") {
        await ctx.db.patch(args.sessionId, {
          status: "active",
          isActive: true,
          expiresAt: undefined,
        });
      }

      // Update active users list
      await ctx.db.patch(args.sessionId, {
        activeUsers: activeUserIds,
        lastActivity: now,
      });
    }

    return {
      success: true,
      activeParticipants,
      sessionStatus: activeParticipants > 0 ? "active" : "scheduled_for_deletion"
    };
  },
});

// Get sessions that are scheduled for deletion and past their expiry time
export const getExpiredSessions = query({
  args: {},
  handler: async (ctx) => {
    const now = Date.now();

    const allSessions = await ctx.db.query("collaborativeSessions").collect();

    const expiredSessions = allSessions.filter(session =>
      session.status === "scheduled_for_deletion" &&
      session.expiresAt &&
      session.expiresAt < now
    );

    return expiredSessions;
  },
});

// Internal function to cleanup expired sessions (called by cron job)
export const cleanupExpiredSessions = internalMutation({
  handler: async (ctx) => {
    const now = Date.now();

    // Find sessions that should be deleted
    const allSessions = await ctx.db.query("collaborativeSessions").collect();

    const expiredSessions = allSessions.filter(session =>
      session.status === "scheduled_for_deletion" &&
      session.expiresAt &&
      session.expiresAt < now
    );

    let cleanedCount = 0;

    for (const session of expiredSessions) {
      try {
        // Delete all participants
        const participants = await ctx.db
          .query("sessionParticipants")
          .withIndex("by_session_id", (q) => q.eq("sessionId", session._id))
          .collect();

        for (const participant of participants) {
          await ctx.db.delete(participant._id);
```

```
        }

        // Delete all code operations
        const operations = await ctx.db
          .query("codeOperations")
          .withIndex("by_session_id", (q) => q.eq("sessionId", session._id))
          .collect();

        for (const operation of operations) {
          await ctx.db.delete(operation._id);
        }

        // Delete all chat messages
        const messages = await ctx.db
          .query("sessionMessages")
          .withIndex("by_session_id", (q) => q.eq("sessionId", session._id))
          .collect();

        for (const message of messages) {
          await ctx.db.delete(message._id);
        }

        // Finally, delete the session
        await ctx.db.delete(session._id);
        cleanedCount++;

      } catch (error) {
        console.error(`Failed to delete session ${session._id}:`, error);
      }
    }

    console.log(`Cleaned up ${cleanedCount} expired collaboration sessions`);
    return { cleanedCount };
  },
});

// Check and update activity for all active sessions (called by cron job)
export const checkAllSessionsActivity = internalMutation({
  handler: async (ctx) => {
    const activeSessions = await ctx.db
      .query("collaborativeSessions")
      .withIndex("by_is_active", (q) => q.eq("isActive", true))
      .collect();

    let processedCount = 0;

    for (const session of activeSessions) {
      try {
        const now = Date.now();

        // Get all participants for this session
        const participants = await ctx.db
          .query("sessionParticipants")
          .withIndex("by_session_id", (q) => q.eq("sessionId", session._id))
          .collect();

        // Count active participants
        let activeParticipants = 0;
        const activeUserIds: string[] = [];

        for (const participant of participants) {
          const isActiveNow = participant.lastSeen
            ? now - participant.lastSeen <= INACTIVE_THRESHOLD
            : now - participant.lastActive <= INACTIVE_THRESHOLD;

          if (participant.isActive !== isActiveNow) {
            await ctx.db.patch(participant._id, {
              isActive: isActiveNow,
              lastActive: isActiveNow ? participant.lastActive : now,
            });
          }

          if (isActiveNow) {
            activeParticipants++;
            activeUserIds.push(participant.userId);
          }
        }

        // Update session based on active participa
```

```
... (truncated 1909 characters)
```

■ File: convex/sessionFiles.ts

Size: 3712 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■■ ***Business Logic*** *- Core application functionality*
> ■ *Contains: 2 imports, 4 exports, ~0 functions/constants*
> ■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { mutation, query } from "./_generated/server";
import { v } from "convex/values";

// Get a specific session file
export const getSessionFile = query({
  args: {
    fileId: v.id("sessionFiles"),
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();
    if (!identity || identity.subject !== args.userId) {
      throw new Error("Unauthorized");
    }

    const file = await ctx.db.get(args.fileId);
    if (!file || file.sessionId !== args.sessionId) {
      throw new Error("File not found");
    }

    return file;
  },
});

// Create a new session file
export const createFile = mutation({
  args: {
    sessionId: v.id("collaborativeSessions"),
    name: v.string(),
    language: v.string(),
    folderId: v.optional(v.id("sessionFolders")),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();
    if (!identity || identity.subject !== args.userId) {
      throw new Error("Unauthorized");
    }

    // Check if session exists
    const session = await ctx.db.get(args.sessionId);
    if (!session) {
      throw new Error("Session not found");
    }

    // Build path
    let path = "";
    if (args.folderId) {
      const folder = await ctx.db.get(args.folderId);
      if (folder) {
        path = `${folder.path}/${args.name}`;
      } else {
        path = `/${args.name}`;
```

```
      }
    } else {
      path = `/${args.name}`;
    }

    // Check for duplicate names in the same folder
    const existingFile = await ctx.db
      .query("sessionFiles")
      .withIndex("by_session_and_folder", (q) =>
        q.eq("sessionId", args.sessionId).eq("folderId", args.folderId)
      )
      .filter((q) => q.eq(q.field("name"), args.name))
      .first();

    if (existingFile) {
      throw new Error(`File "${args.name}" already exists`);
    }

    const now = Date.now();
    return await ctx.db.insert("sessionFiles", {
      sessionId: args.sessionId,
      userId: args.userId,
      folderId: args.folderId,
      name: args.name,
      language: args.language,
      code: `// ${args.name}\n// Created in session: ${session.name}\n\n`,
      createdAt: now,
      updatedAt: now,
      path,
    });
  },
});

// Update session file content
export const updateFile = mutation({
  args: {
    fileId: v.id("sessionFiles"),
    sessionId: v.id("collaborativeSessions"),
    code: v.string(),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();
    if (!identity || identity.subject !== args.userId) {
      throw new Error("Unauthorized");
    }

    const file = await ctx.db.get(args.fileId);
    if (!file || file.sessionId !== args.sessionId) {
      throw new Error("File not found");
    }

    await ctx.db.patch(args.fileId, {
      code: args.code,
      updatedAt: Date.now(),
    });
  },
});

// Delete a session file
export const deleteFile = mutation({
  args: {
    fileId: v.id("sessionFiles"),
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();
    if (!identity || identity.subject !== args.userId) {
      throw new Error("Unauthorized");
    }

    const file = await ctx.db.get(args.fileId);
    if (!file || file.sessionId !== args.sessionId) {
      throw new Error("File not found");
    }

    await ctx.db.delete(args.fileId);
  },
```

```
});
```

■ File: convex/sessionFolders.ts

Size: 4650 bytes | Last Modified: 2025-10-31 16:49:43

■ ***File Overview:***
■■ ***Business Logic*** *- Core application functionality*
■ *Contains: 2 imports, 3 exports, ~2 functions/constants*
■ *Uses Convex backend framework*

■ Functions & Logic Analysis:

Function: deleteRecursively
• Asynchronous function (uses async/await)
• Parameters: folderId
• Fetches data from the database

```
  const deleteRecursively = async (folderId: string) => {
    // Get all child folders
    const childFolders = await ctx.db
      .query("sessionFolders")
      .withIndex("by_parent_folder", (q) => q.eq("parentFolderId", folderId as any))
      .collect();

    // Recursively delete child folders
    for (const childFolder of childFolders) {
      await deleteRecursively(childFolder._id);
    }

    // Delete all files in this folder
    const files = await ctx.db
   ...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { mutation, query } from "./_generated/server";
import { v } from "convex/values";

// Get folder tree for a specific session
export const getSessionFolderTree = query({
  args: {
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();
    if (!identity || identity.subject !== args.userId) {
      throw new Error("Unauthorized");
    }

    // Get all folders for this session
    const folders = await ctx.db
      .query("sessionFolders")
      .withIndex("by_session_id", (q) => q.eq("sessionId", args.sessionId))
      .collect();

    // Get all files for this session
    const files = await ctx.db
      .query("sessionFiles")
      .withIndex("by_session_id", (q) => q.eq("sessionId", args.sessionId))
      .collect();
```

```
    // Build nested structure
    const buildFolderTree = (parentId?: string): any[] => {
      const childFolders = folders
        .filter((folder) => folder.parentFolderId === parentId)
        .map((folder) => ({
          ...folder,
          children: buildFolderTree(folder._id),
          files: files.filter((file) => file.folderId === folder._id),
        }));
      return childFolders;
    };

    return {
      folders: buildFolderTree(),
      files: files.filter((file) => !file.folderId), // Root files
    };
  },
});

// Create a new folder
export const createFolder = mutation({
  args: {
    sessionId: v.id("collaborativeSessions"),
    name: v.string(),
    parentFolderId: v.optional(v.id("sessionFolders")),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();
    if (!identity || identity.subject !== args.userId) {
      throw new Error("Unauthorized");
    }

    // Check if session exists and user has access
    const session = await ctx.db.get(args.sessionId);
    if (!session) {
      throw new Error("Session not found");
    }

    // Build path
    let path = "";
    if (args.parentFolderId) {
      const parentFolder = await ctx.db.get(args.parentFolderId);
      if (parentFolder) {
        path = `${parentFolder.path}/${args.name}`;
      } else {
        path = `/${args.name}`;
      }
    } else {
      path = `/${args.name}`;
    }

    // Check for duplicate names in the same parent
    const existingFolder = await ctx.db
      .query("sessionFolders")
      .withIndex("by_session_and_parent", (q) =>
        q.eq("sessionId", args.sessionId).eq("parentFolderId", args.parentFolderId)
      )
      .filter((q) => q.eq(q.field("name"), args.name))
      .first();

    if (existingFolder) {
      throw new Error(`Folder "${args.name}" already exists`);
    }

    return await ctx.db.insert("sessionFolders", {
      sessionId: args.sessionId,
      userId: args.userId,
      name: args.name,
      parentFolderId: args.parentFolderId,
      createdAt: Date.now(),
      path,
    });
  },
});

// Delete a folder and all its contents
export const deleteFolder = mutation({
  args: {
```

```
    folderId: v.id("sessionFolders"),
    sessionId: v.id("collaborativeSessions"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const identity = await ctx.auth.getUserIdentity();
    if (!identity || identity.subject !== args.userId) {
      throw new Error("Unauthorized");
    }

    // Check if folder exists and belongs to session
    const folder = await ctx.db.get(args.folderId);
    if (!folder || folder.sessionId !== args.sessionId) {
      throw new Error("Folder not found");
    }

    // Recursively delete all subfolders and files
    const deleteRecursively = async (folderId: string) => {
      // Get all child folders
      const childFolders = await ctx.db
        .query("sessionFolders")
        .withIndex("by_parent_folder", (q) => q.eq("parentFolderId", folderId as any))
        .collect();

      // Recursively delete child folders
      for (const childFolder of childFolders) {
        await deleteRecursively(childFolder._id);
      }

      // Delete all files in this folder
      const files = await ctx.db
        .query("sessionFiles")
        .withIndex("by_folder_id", (q) => q.eq("folderId", folderId as any))
        .collect();

      for (const file of files) {
        await ctx.db.delete(file._id);
      }

      // Delete the folder itself
      await ctx.db.delete(folderId as any);
    };

    await deleteRecursively(args.folderId);
  },
});
```

■ File: convex/files.ts

Size: 11622 bytes | Last Modified: 2025-10-31 16:49:43

■ ***File Overview:***
■■ ***Business Logic*** *- Core application functionality*
■ *Contains: 2 imports, 11 exports, ~3 functions/constants*
■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { mutation, query } from "./_generated/server";
import { v } from "convex/values";

// Create a new file
export const createFile = mutation({
  args: {
    name: v.string(),
    language: v.string(),
    code: v.optional(v.string()),
```

```
    description: v.optional(v.string()),
    folderId: v.optional(v.id("folders")),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { name, language, code, description, folderId, userId } = args;

    // Calculate the path
    let path = "";
    if (folderId) {
      const folder = await ctx.db.get(folderId);
      if (!folder) {
        throw new Error("Folder not found");
      }
      if (folder.userId !== userId) {
        throw new Error("Permission denied");
      }
      path = `${folder.path}/${name}`;
    } else {
      path = `/${name}`;
    }

    // Check if file with same name exists in same location
    const existingFile = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_and_folder", (q) =>
        q.eq("userId", userId).eq("folderId", folderId)
      )
      .filter((q) => q.eq(q.field("name"), name))
      .first();

    if (existingFile) {
      throw new Error("File with this name already exists in this location");
    }

    const now = Date.now();
    const fileId = await ctx.db.insert("practiceFiles", {
      userId,
      folderId,
      name,
      language,
      code: code || getDefaultCodeForLanguage(language),
      description,
      isShared: false,
      createdAt: now,
      updatedAt: now,
      path,
    });

    return fileId;
  },
});

// Get user's files
export const getUserFiles = query({
  args: {
    userId: v.string(),
    folderId: v.optional(v.id("folders")),
  },
  handler: async (ctx, args) => {
    const files = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_and_folder", (q) =>
        q.eq("userId", args.userId).eq("folderId", args.folderId)
      )
      .collect();

    return files.sort((a, b) => b.updatedAt - a.updatedAt);
  },
});

// Get file by ID
export const getFile = query({
  args: {
    fileId: v.id("practiceFiles"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const file = await ctx.db.get(args.fileId);
```

```
    if (!file) {
      throw new Error("File not found");
    }

    if (file.userId !== args.userId && !file.isShared) {
      throw new Error("Permission denied");
    }

    return file;
  },
});

// Update file content
export const updateFile = mutation({
  args: {
    fileId: v.id("practiceFiles"),
    code: v.optional(v.string()),
    description: v.optional(v.string()),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { fileId, code, description, userId } = args;

    const file = await ctx.db.get(fileId);
    if (!file) {
      throw new Error("File not found");
    }

    if (file.userId !== userId) {
      throw new Error("Permission denied");
    }

    const updateData: {
      updatedAt: number;
      code?: string;
      description?: string;
    } = {
      updatedAt: Date.now(),
    };

    if (code !== undefined) {
      updateData.code = code;
    }

    if (description !== undefined) {
      updateData.description = description;
    }

    await ctx.db.patch(fileId, updateData);

    return { success: true };
  },
});

// Rename file
export const renameFile = mutation({
  args: {
    fileId: v.id("practiceFiles"),
    newName: v.string(),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { fileId, newName, userId } = args;

    const file = await ctx.db.get(fileId);
    if (!file) {
      throw new Error("File not found");
    }

    if (file.userId !== userId) {
      throw new Error("Permission denied");
    }

    // Check if file with same name exists in same location
    const existingFile = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_and_folder", (q) =>
        q.eq("userId", userId).eq("folderId", file.folderId)
      )
```

```
      .filter((q) =>
        q.and(
          q.eq(q.field("name"), newName),
          q.neq(q.field("_id"), fileId)
        )
      )
      .first();

    if (existingFile) {
      throw new Error("File with this name already exists in this location");
    }

    // Update path
    const pathParts = file.path.split('/');
    pathParts[pathParts.length - 1] = newName;
    const newPath = pathParts.join('/');

    await ctx.db.patch(fileId, {
      name: newName,
      path: newPath,
      updatedAt: Date.now(),
    });

    return { success: true };
  },
});

// Delete file
export const deleteFile = mutation({
  args: {
    fileId: v.id("practiceFiles"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { fileId, userId } = args;

    const file = await ctx.db.get(fileId);
    if (!file) {
      throw new Error("File not found");
    }

    if (file.userId !== userId) {
      throw new Error("Permission denied");
    }

    await ctx.db.delete(fileId);

    return { success: true };
  },
});

// Move file to different folder
export const moveFile = mutation({
  args: {
    fileId: v.id("practiceFiles"),
    targetFolderId: v.optional(v.id("folders")),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { fileId, targetFolderId, userId } = args;

    const file = await ctx.db.get(fileId);
    if (!file) {
      throw new Error("File not found");
    }

    if (file.userId !== userId) {
      throw new Error("Permission denied");
    }

    // Validate target folder
    if (targetFolderId) {
      const targetFolder = await ctx.db.get(targetFolderId);
      if (!targetFolder) {
        throw new Error("Target folder not found");
      }
      if (targetFolder.userId !== userId) {
        throw new Error("Permission denied");
      }
```

```
    }

    // Check if file with same name exists in target location
    const existingFile = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_and_folder", (q) =>
        q.eq("userId", userId).eq("folderId", targetFolderId)
      )
      .filter((q) =>
        q.and(
          q.eq(q.field("name"), file.name),
          q.neq(q.field("_id"), fileId)
        )
      )
      .first();

    if (existingFile) {
      throw new Error("File with this name already exists in target location");
    }

    // Calculate new path
    let newPath = "";
    if (targetFolderId) {
      const targetFolder = await ctx.db.get(targetFolderId);
      newPath = `${targetFolder!.path}/${file.name}`;
    } else {
      newPath = `/${file.name}`;
    }

    await ctx.db.patch(fileId, {
      folderId: targetFolderId,
      path: newPath,
      updatedAt: Date.now(),
    });

    return { success: true };
  },
});

// Search files
export const searchFiles = query({
  args: {
    userId: v.string(),
    searchTerm: v.string(),
  },
  handler: async (ctx, args) => {
    const { userId, searchTerm } = args;

    const files = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_id", (q) => q.eq("userId", userId))
      .collect();

    // Filter files that match search term in name, description, or code
    const filteredFiles = files.filter(file =>
      file.name.toLowerCase().includes(searchTerm.toLowerCase()) ||
      (file.description && file.description.toLowerCase().includes(searchTerm.toLowerCase())) ||
      file.code.toLowerCase().includes(searchTerm.toLowerCase())
    );

    return filteredFiles.sort((a, b) => b.updatedAt - a.updatedAt);
  },
});

// Get recent files
export const getRecentFiles = query({
  args: {
    userId: v.string(),
    limit: v.optional(v.number()),
  },
  handler: async (ctx, args) => {
    const { userId, limit = 10 } = args;

    const files = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_id", (q) => q.eq("userId", userId))
      .collect();

    return files
```

```
      .sort((a, b) => b.updatedAt - a.updatedAt)
      .slice(0, limit);
  },
});

// Get file statistics
exp

... (truncated 3206 characters)
```

■ File: convex/folders.ts

Size: 7001 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■■ ***Business Logic*** *- Core application functionality*
> ■ *Contains: 2 imports, 6 exports, ~2 functions/constants*
> ■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { mutation, query } from "./_generated/server";
import { v } from "convex/values";

// Create a new folder
export const createFolder = mutation({
  args: {
    name: v.string(),
    parentFolderId: v.optional(v.id("folders")),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { name, parentFolderId, userId } = args;

    // Calculate the path
    let path = "";
    if (parentFolderId) {
      const parentFolder = await ctx.db.get(parentFolderId);
      if (!parentFolder) {
        throw new Error("Parent folder not found");
      }
      if (parentFolder.userId !== userId) {
        throw new Error("Permission denied");
      }
      path = `${parentFolder.path}/${name}`;
    } else {
      path = `/${name}`;
    }

    // Check if folder with same name exists in same location
    const existingFolder = await ctx.db
      .query("folders")
      .withIndex("by_user_id", (q) => q.eq("userId", userId))
      .filter((q) =>
        q.and(
          q.eq(q.field("name"), name),
          parentFolderId
            ? q.eq(q.field("parentFolderId"), parentFolderId)
            : q.eq(q.field("parentFolderId"), undefined)
        )
      )
      .first();

    if (existingFolder) {
      throw new Error("Folder with this name already exists");
    }
```

```
    const folderId = await ctx.db.insert("folders", {
      userId,
      name,
      parentFolderId,
      isShared: false,
      createdAt: Date.now(),
      path,
    });

    return folderId;
  },
});

// Get user's folders
export const getUserFolders = query({
  args: {
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const folders = await ctx.db
      .query("folders")
      .withIndex("by_user_id", (q) => q.eq("userId", args.userId))
      .collect();

    return folders;
  },
});

// Get folders by parent
export const getFoldersByParent = query({
  args: {
    userId: v.string(),
    parentFolderId: v.optional(v.id("folders")),
  },
  handler: async (ctx, args) => {
    const folders = await ctx.db
      .query("folders")
      .withIndex("by_parent_folder", (q) =>
        args.parentFolderId
          ? q.eq("parentFolderId", args.parentFolderId)
          : q.eq("parentFolderId", undefined)
      )
      .filter((q) => q.eq(q.field("userId"), args.userId))
      .collect();

    return folders;
  },
});

// Rename folder
export const renameFolder = mutation({
  args: {
    folderId: v.id("folders"),
    newName: v.string(),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { folderId, newName, userId } = args;

    const folder = await ctx.db.get(folderId);
    if (!folder) {
      throw new Error("Folder not found");
    }

    if (folder.userId !== userId) {
      throw new Error("Permission denied");
    }

    // Update path
    const oldPath = folder.path;
    const pathParts = oldPath.split('/');
    pathParts[pathParts.length - 1] = newName;
    const newPath = pathParts.join('/');

    await ctx.db.patch(folderId, {
      name: newName,
      path: newPath,
    });
```

```
    // Update all subfolders and files paths
    const allFolders = await ctx.db
      .query("folders")
      .withIndex("by_user_id", (q) => q.eq("userId", userId))
      .collect();

    const allFiles = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_id", (q) => q.eq("userId", userId))
      .collect();

    // Update subfolder paths
    for (const subfolder of allFolders) {
      if (subfolder.path.startsWith(oldPath + '/')) {
        const newSubfolderPath = subfolder.path.replace(oldPath, newPath);
        await ctx.db.patch(subfolder._id, {
          path: newSubfolderPath,
        });
      }
    }

    // Update file paths
    for (const file of allFiles) {
      if (file.path.startsWith(oldPath + '/')) {
        const newFilePath = file.path.replace(oldPath, newPath);
        await ctx.db.patch(file._id, {
          path: newFilePath,
        });
      }
    }

    return { success: true };
  },
});

// Delete folder and all its contents
export const deleteFolder = mutation({
  args: {
    folderId: v.id("folders"),
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const { folderId, userId } = args;

    const folder = await ctx.db.get(folderId);
    if (!folder) {
      throw new Error("Folder not found");
    }

    if (folder.userId !== userId) {
      throw new Error("Permission denied");
    }

    // Get all subfolders
    const subfolders = await ctx.db
      .query("folders")
      .withIndex("by_user_id", (q) => q.eq("userId", userId))
      .filter((q) => q.gte(q.field("path"), folder.path + '/'))
      .filter((q) => q.lt(q.field("path"), folder.path + '0'))
      .collect();

    // Get all files in this folder and subfolders
    const files = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_id", (q) => q.eq("userId", userId))
      .filter((q) => q.gte(q.field("path"), folder.path + '/'))
      .filter((q) => q.lt(q.field("path"), folder.path + '0'))
      .collect();

    // Also get files directly in this folder
    const directFiles = await ctx.db
      .query("practiceFiles")
      .withIndex("by_folder_id", (q) => q.eq("folderId", folderId))
      .collect();

    // Delete all files
    for (const file of [...files, ...directFiles]) {
      await ctx.db.delete(file._id);
    }
```

```
    // Delete all subfolders
    for (const subfolder of subfolders) {
      await ctx.db.delete(subfolder._id);
    }

    // Delete the folder itself
    await ctx.db.delete(folderId);

    return { success: true };
  },
});

// Get folder tree structure
export const getFolderTree = query({
  args: {
    userId: v.string(),
  },
  handler: async (ctx, args) => {
    const folders = await ctx.db
      .query("folders")
      .withIndex("by_user_id", (q) => q.eq("userId", args.userId))
      .collect();

    const files = await ctx.db
      .query("practiceFiles")
      .withIndex("by_user_id", (q) => q.eq("userId", args.userId))
      .collect();

    // Build tree structure
    // eslint-disable-next-line @typescript-eslint/no-explicit-any
    const buildTree = (parentId: string | undefined): any[] => {
      const children = folders
        .filter(f => f.parentFolderId === parentId)
        .map(folder => ({
          ...folder,
          type: 'folder' as const,
          children: buildTree(folder._id),
          files: files.filter(f => f.folderId === folder._id),
        }));

      return children;
    };

    // Get root folders and files
    const rootFolders = buildTree(undefined);
    const rootFiles = files.filter(f => !f.folderId);

    return {
      folders: rootFolders,
      files: rootFiles,
    };
  },
});
```

■ File: convex/snippets.ts

Size: 6809 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■ *Contains: 2 imports, 11 exports, ~1 functions/constants*
> ■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { v } from "convex/values";
import { mutation, query } from "./_generated/server";

export const createSnippet = mutation({
```

```
    args: {
        title: v.string(),
        language: v.string(),
        code: v.string(),
    },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) throw new Error("Not authenticated");

        const user = await ctx.db
            .query("users")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), identity.subject))
            .first();

        if (!user) throw new Error("User not found");

        const snippetId = await ctx.db.insert("snippets", {
            userId: identity.subject,
            userName: user.name,
            title: args.title,
            language: args.language,
            code: args.code,
        });

        return snippetId;
    },
});

export const deleteSnippet = mutation({
    args: {
        snippetId: v.id("snippets"),
    },

    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) throw new Error("Not authenticated");

        const snippet = await ctx.db.get(args.snippetId);
        if (!snippet) throw new Error("Snippet not found");

        if (snippet.userId !== identity.subject) {
            throw new Error("Not authorized to delete this snippet");
        }

        const comments = await ctx.db
            .query("snippetComments")
            .withIndex("by_snippet_id")
            .filter((q) => q.eq(q.field("snippetId"), args.snippetId))
            .collect();

        for (const comment of comments) {
            await ctx.db.delete(comment._id);
        }

        const stars = await ctx.db
            .query("stars")
            .withIndex("by_snippet_id")
            .filter((q) => q.eq(q.field("snippetId"), args.snippetId))
            .collect();

        for (const star of stars) {
            await ctx.db.delete(star._id);
        }

        await ctx.db.delete(args.snippetId);
    },
});

export const starSnippet = mutation({
    args: {
        snippetId: v.id("snippets"),
    },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) throw new Error("Not authenticated");

        const existing = await ctx.db
            .query("stars")
```

```
            .withIndex("by_user_id_and_snippet_id")
            .filter(
                (q) =>
                    q.eq(q.field("userId"), identity.subject) && q.eq(q.field("snippetId"), args.snippetId)
            )
            .first();

        if (existing) {
            await ctx.db.delete(existing._id);
        } else {
            await ctx.db.insert("stars", {
                userId: identity.subject,
                snippetId: args.snippetId,
            });
        }
    },
});

export const addComment = mutation({
    args: {
        snippetId: v.id("snippets"),
        content: v.string(),
    },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) throw new Error("Not authenticated");

        const user = await ctx.db
            .query("users")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), identity.subject))
            .first();

        if (!user) throw new Error("User not found");

        return await ctx.db.insert("snippetComments", {
            snippetId: args.snippetId,
            userId: identity.subject,
            userName: user.name,
            content: args.content,
        });
    },
});

export const deleteComment = mutation({
    args: { commentId: v.id("snippetComments") },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) throw new Error("Not authenticated");

        const comment = await ctx.db.get(args.commentId);
        if (!comment) throw new Error("Comment not found");

        // Check if the user is the comment author
        if (comment.userId !== identity.subject) {
            throw new Error("Not authorized to delete this comment");
        }

        await ctx.db.delete(args.commentId);
    },
});

export const getSnippets = query({
    handler: async (ctx) => {
        const snippets = await ctx.db.query("snippets").order("desc").collect();
        return snippets;
    },
});

export const getSnippetById = query({
    args: { snippetId: v.id("snippets") },
    handler: async (ctx, args) => {
        const snippet = await ctx.db.get(args.snippetId);
        if (!snippet) throw new Error("Snippet not found");

        return snippet;
    },
});
```

```
export const getComments = query({
    args: { snippetId: v.id("snippets") },
    handler: async (ctx, args) => {
        const comments = await ctx.db
            .query("snippetComments")
            .withIndex("by_snippet_id")
            .filter((q) => q.eq(q.field("snippetId"), args.snippetId))
            .order("desc")
            .collect();

        return comments;
    },
});

export const isSnippetStarred = query({
    args: {
        snippetId: v.id("snippets"),
    },
    handler: async (ctx, args) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) return false;

        const star = await ctx.db
            .query("stars")
            .withIndex("by_user_id_and_snippet_id")
            .filter(
                (q) =>
                    q.eq(q.field("userId"), identity.subject) && q.eq(q.field("snippetId"), args.snippetId)
            )
            .first();

        return !!star;
    },
});

export const getSnippetStarCount = query({
    args: { snippetId: v.id("snippets") },
    handler: async (ctx, args) => {
        const stars = await ctx.db
            .query("stars")
            .withIndex("by_snippet_id")
            .filter((q) => q.eq(q.field("snippetId"), args.snippetId))
            .collect();

        return stars.length;
    },
});

export const getStarredSnippets = query({
    handler: async (ctx) => {
        const identity = await ctx.auth.getUserIdentity();
        if (!identity) return [];

        const stars = await ctx.db
            .query("stars")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), identity.subject))
            .collect();

        const snippets = await Promise.all(stars.map((star) => ctx.db.get(star.snippetId)));

        return snippets.filter((snippet) => snippet !== null);
    },
});
```

■ File: convex/users.ts

Size: 1879 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■ *Contains: 2 imports, 3 exports, ~0 functions/constants*

■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { v } from "convex/values";
import { mutation, query } from "./_generated/server";


export const syncUser = mutation({
    args: {
        userId: v.string(),
        email: v.string(),
        name: v.string(),
    },
    handler: async (ctx, args) => {
        const existingUser = await ctx.db
            .query("users")
            .filter((q) => q.eq(q.field("userId"), args.userId))
            .first();

        if (!existingUser) {
            await ctx.db.insert("users", {
                userId: args.userId,
                email: args.email,
                name: args.name,
                isPro: false,
            });
        }
    }
})

export const getUser = query({
    args: { userId: v.string() },

    handler: async (ctx, args) => {
        if (!args.userId) return null;

        const user = await ctx.db
            .query("users")
            .withIndex("by_user_id")
            .filter((q) => q.eq(q.field("userId"), args.userId))
            .first();

        if (!user) return null;

        return user;
    },
});

export const upgradeToPro = mutation({
    args: {
        email: v.string(),
        lemonSqueezyCustomerId: v.string(),
        lemonSqueezyOrderId: v.string(),
        amount: v.number(),
    },
    handler: async (ctx, args) => {
        const user = await ctx.db
            .query("users")
            .filter((q) => q.eq(q.field("email"), args.email))
            .first();

        if (!user) throw new Error("User not found");

        await ctx.db.patch(user._id, {
            isPro: true,
            proSince: Date.now(),
            lemonSqueezyCustomerId: args.lemonSqueezyCustomerId,
            lemonSqueezyOrderId: args.lemonSqueezyOrderId,
        });

        return { success: true };
    },
});
```

■ File: convex/lemonSqueezy.ts

Size: 875 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■ *Contains: 3 imports, 1 exports, ~1 functions/constants*
> ■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
"use node";
import { v } from "convex/values";
import { internalAction } from "./_generated/server";
import { createHmac } from "crypto";

const webhookSecret = process.env.LEMON_SQUEEZY_WEBHOOK_SECRET!;

function verifySignature(payload: string, signature: string): boolean {
    const hmac = createHmac("sha256", webhookSecret);
    const computedSignature = hmac.update(payload).digest("hex");
    return signature === computedSignature;
}

export const verifyWebhook = internalAction({
    args: {
        payload: v.string(),
        signature: v.string(),
    },
    handler: async (ctx, args) => {
        const isValid = verifySignature(args.payload, args.signature);

        if (!isValid) {
            throw new Error("Invalid signature");
        }

        return JSON.parse(args.payload);
    },
});
```

■ File: convex/http.ts

Size: 3601 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■ *Contains: 5 imports, 1 exports, ~0 functions/constants*
> ■ *Uses Convex backend framework*
> ■ *Integrates Clerk authentication*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { httpRouter } from "convex/server";
import { httpAction } from "./_generated/server";
import { Webhook } from "svix";
import { WebhookEvent } from "@clerk/nextjs/server";
import { api, internal } from "./_generated/api";
```

```
const http = httpRouter();

http.route({
    path: "/lemon-squeezy-webhook",
    method: "POST",
    handler: httpAction(async (ctx, request) => {
        const payloadString = await request.text();
        const signature = request.headers.get("X-Signature");

        if (!signature) {
            return new Response("Missing X-Signature header", { status: 400 });
        }

        try {
            const payload = await ctx.runAction(internal.lemonSqueezy.verifyWebhook, {
                payload: payloadString,
                signature,
            });

            if (payload.meta.event_name === "order_created") {
                const { data } = payload;

                const { success } = await ctx.runMutation(api.users.upgradeToPro, {
                    email: data.attributes.user_email,
                    lemonSqueezyCustomerId: data.attributes.customer_id.toString(),
                    lemonSqueezyOrderId: data.id,
                    amount: data.attributes.total,
                });

                if (success) {
                    // Something
                }
            }

            return new Response("Webhook processed successfully", { status: 200 });
        } catch (error) {
            console.log("Webhook error:", error);
            return new Response("Error processing webhook", { status: 500 });
        }
    }),
});

http.route({
    path: "/clerk-webhook",
    method: "POST",

    handler: httpAction(async (context, request) => {
        const webHookSecret = process.env.CLERK_WEBHOOK_SECRET

        if (!webHookSecret) {
            throw new Error("CLERK_WEBHOOK_SECRET is not set");
        }

        const svix_id = request.headers.get("svix-id");
        const svix_signature = request.headers.get("svix-signature");
        const svix_timestamp = request.headers.get("svix-timestamp");

        if (!svix_id || !svix_signature || !svix_timestamp) {
            throw new Error("Missing svix headers");
        }

        const payload = await request.json();
        const body = JSON.stringify(payload);

        let wh = new Webhook(webHookSecret);

        let evt: WebhookEvent;

        try {
            evt = wh.verify(body, {
                "svix-id": svix_id,
                "svix-timestamp": svix_timestamp,
                "svix-signature": svix_signature,
            }) as WebhookEvent;
        } catch (err) {
            console.error("Error verifying webhook:", err);
            return new Response("Error occurred", { status: 400 });
        }
```

```
        const eventType = evt.type;

        if (eventType === "user.created") {
            const { id, email_addresses, first_name, last_name } = evt.data;

            const email = email_addresses[0].email_address;
            const name = `${first_name || ""} ${last_name || ""}`.trim();

            try {

                await context.runMutation(api.users.syncUser, {
                    userId: id,
                    email,
                    name
                })

            } catch (error: any) {
                return new Response(error.message, { status: 400 });
            }
        }

        return new Response("OK", { status: 200 });
    })
})

export default http;
```

■ File: convex/crons.ts

Size: 560 bytes | Last Modified: 2025-10-31 16:49:43

> ■ ***File Overview:***
> ■ *Contains: 2 imports, 1 exports, ~0 functions/constants*
> ■ *Uses Convex backend framework*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { cronJobs } from "convex/server";
import { internal } from "./_generated/api";

const crons = cronJobs();

// Run every 10 minutes to check session activity and cleanup expired sessions
crons.interval(
  "session-activity-check",
  { minutes: 10 },
  internal.sessionActivity.checkAllSessionsActivity
);

// Run every 30 minutes to cleanup expired sessions
crons.interval(
  "cleanup-expired-sessions",
  { minutes: 30 },
  internal.sessionActivity.cleanupExpiredSessions
);

export default crons;
```

■ File: convex/migration.ts

Size: 1440 bytes | Last Modified: 2025-10-31 16:49:43

■ ***File Overview:***
■ *Contains: 1 imports, 1 exports, ~0 functions/constants*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 2
import { internalMutation } from "./_generated/server";

// Migration script to update existing sessions with new fields
export const migrateExistingSessions = internalMutation({
  handler: async (ctx) => {
    const sessions = await ctx.db.query("collaborativeSessions").collect();
    let updatedCount = 0;

    for (const session of sessions) {
      const updates: Record<string, unknown> = {};

      // Add status field if missing
      if (!session.status) {
        updates.status = session.isActive ? "active" : "inactive";
      }

      // Only update if there are changes
      if (Object.keys(updates).length > 0) {
        await ctx.db.patch(session._id, updates);
        updatedCount++;
      }
    }

    // Update participants with lastSeen field
    const participants = await ctx.db.query("sessionParticipants").collect();
    let participantUpdatedCount = 0;

    for (const participant of participants) {
      if (!participant.lastSeen) {
        await ctx.db.patch(participant._id, {
          lastSeen: participant.lastActive,
        });
        participantUpdatedCount++;
      }
    }

    console.log(`Migration completed: Updated ${updatedCount} sessions and ${participantUpdatedCount} partic
    return {
      updatedSessions: updatedCount,
      updatedParticipants: participantUpdatedCount
    };
  },
});
```

Phase 3: Server-Side Logic, Middleware & API Routes

This phase documents server-side middleware, authentication checks, request/response handling, and API route implementations.

■ File: src/middleware.ts

Size: 463 bytes | Last Modified: 2025-10-31 16:51:09

■ ***File Overview:***
■ ***Middleware*** *- Intercepts requests for authentication/logging*
■ *Contains: 1 imports, 2 exports, ~0 functions/constants*
■ *Integrates Clerk authentication*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 3
import { clerkMiddleware } from "@clerk/nextjs/server";

export default clerkMiddleware();

export const config = {
    matcher: [
        // Skip Next.js internals and all static files, unless found in search params
        '/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip
        // Always run for API routes
        '/(api|trpc)(.*)',
    ],
};
```

■ File: src/app/api/session-leave/route.ts

Size: 1055 bytes | Last Modified: 2025-10-31 16:51:09

■ ***File Overview:***
■■ ***API Route Handler*** *- Handles HTTP requests and responses*
■ *Contains: 4 imports, 1 exports, ~1 functions/constants*
■ *Uses Convex backend framework*

■ Functions & Logic Analysis:

Function: POST
• Asynchronous function (uses async/await)
• Parameters: request
• Performs database mutations (create/update/delete operations)
• Includes error handling

```
export async function POST(request: NextRequest) {
  try {
    const { sessionId, userId } = await request.json();

    if (!sessionId || !userId) {
```

```
      return NextResponse.json(
        { error: 'Missing sessionId or userId' },
        { status: 400 }
      );
    }

    // Call the leave session mutation
    await convex.mutation(api.collaboration.leaveSession, {
      sessionId: sessionId as Id<"collaborativeSessions">,
      userId,
    });

    return NextResponse.json({ success: true });...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 3
import { NextRequest, NextResponse } from 'next/server';
import { ConvexHttpClient } from 'convex/browser';
import { api } from '../../../../convex/_generated/api';
import { Id } from '../../../../convex/_generated/dataModel';

const convex = new ConvexHttpClient(process.env.NEXT_PUBLIC_CONVEX_URL!);

export async function POST(request: NextRequest) {
  try {
    const { sessionId, userId } = await request.json();

    if (!sessionId || !userId) {
      return NextResponse.json(
        { error: 'Missing sessionId or userId' },
        { status: 400 }
      );
    }

    // Call the leave session mutation
    await convex.mutation(api.collaboration.leaveSession, {
      sessionId: sessionId as Id<"collaborativeSessions">,
      userId,
    });

    return NextResponse.json({ success: true });
  } catch (error) {
    console.error('Failed to leave session:', error);
    return NextResponse.json(
      { error: 'Failed to leave session' },
      { status: 500 }
    );
  }
}
```

Phase 4: Application Logic from Frontend Files

This phase extracts business logic, state management, event handlers, data processing functions, and utility code from frontend component files. JSX rendering code and pure UI elements are excluded.

■ File: src/store/useCodeEditorStore.ts

Size: 5449 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■■ ***State Management Store*** *- Manages application state using Zustand/Redux*
■ *Contains: 4 imports, 2 exports, ~3 functions/constants*
■ *Uses Zustand for state management*
■ *Integrates Monaco code editor*

■ Functions & Logic Analysis:

Function: getInitialState
• Returns data/object/JSX

```
const getInitialState = () => {
    // if we're on the server, return default values
    if (typeof window === "undefined") {
        return {
            language: "javascript",
            fontSize: 16,
            theme: "vs-dark",
        };
    }

    // if we're on the client, return values from local storage bc localStorage is a browser API.
    const savedLanguage = localStorage.getItem("editor-language") || "javascript";
    const savedTheme = localStorage.getItem("editor-theme") || "vs...
```

Function: getExecutionResult

```
export const getExecutionResult = () => useCodeEditorStore.getState().executionResult;
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { CodeEditorState } from "./../types/index";
import { LANGUAGE_CONFIG } from "@/app/(root)/_constants";
import { create } from "zustand";
import { Monaco } from "@monaco-editor/react";

const getInitialState = () => {
    // if we're on the server, return default values
    if (typeof window === "undefined") {
        return {
            language: "javascript",
            fontSize: 16,
            theme: "vs-dark",
        };
    }

    // if we're on the client, return values from local storage bc localStorage is a browser API.
    const savedLanguage = localStorage.getItem("editor-language") || "javascript";
    const savedTheme = localStorage.getItem("editor-theme") || "vs-dark";
```

```
    const savedFontSize = localStorage.getItem("editor-font-size") || 16;

    return {
        language: savedLanguage,
        theme: savedTheme,
        fontSize: Number(savedFontSize),
    };
};

export const useCodeEditorStore = create<CodeEditorState>((set, get) => {
    const initialState = getInitialState();

    return {
        ...initialState,
        output: "",
        isRunning: false,
        error: null,
        editor: null,
        executionResult: null,

        getCode: () => get().editor?.getValue() || "",

        setEditor: (editor: Monaco) => {
            const savedCode = localStorage.getItem(`editor-code-${get().language}`);
            if (savedCode) editor.setValue(savedCode);

            set({ editor });
        },

        setTheme: (theme: string) => {
            localStorage.setItem("editor-theme", theme);
            set({ theme });
        },

        setFontSize: (fontSize: number) => {
            localStorage.setItem("editor-font-size", fontSize.toString());
            set({ fontSize });
        },

        setLanguage: (language: string) => {
            // Save current language code before switching
            const currentCode = get().editor?.getValue();
            if (currentCode) {
                localStorage.setItem(`editor-code-${get().language}`, currentCode);
            }

            localStorage.setItem("editor-language", language);

            set({
                language,
                output: "",
                error: null,
            });
        },

        runCode: async () => {
            const { language, getCode } = get();
            const code = getCode();

            if (!code) {
                set({ error: "Please enter some code" });
                return;
            }

            set({ isRunning: true, error: null, output: "" });

            try {
                const runtime = LANGUAGE_CONFIG[language].pistonRuntime;
                const response = await fetch("https://emkc.org/api/v2/piston/execute", {
                    method: "POST",
                    headers: {
                        "Content-Type": "application/json",
                    },
                    body: JSON.stringify({
                        language: runtime.language,
                        version: runtime.version,
                        files: [{ content: code }],
                    }),
                });
```

```
                const data = await response.json();

                console.log("data back from piston:", data);

                // handle API-level errors
                if (data.message) {
                    set({ error: data.message, executionResult: { code, output: "", error: data.message } });
                    return;
                }

                // handle compilation errors
                if (data.compile && data.compile.code !== 0) {
                    const error = data.compile.stderr || data.compile.output;
                    set({
                        error,
                        executionResult: {
                            code,
                            output: "",
                            error,
                        },
                    });
                    return;
                }

                if (data.run && data.run.code !== 0) {
                    const error = data.run.stderr || data.run.output;
                    set({
                        error,
                        executionResult: {
                            code,
                            output: "",
                            error,
                        },
                    });
                    return;
                }

                // if we get here, execution was successful
                const output = data.run.output;

                set({
                    output: output.trim(),
                    error: null,
                    executionResult: {
                        code,
                        output: output.trim(),
                        error: null,
                    },
                });
            } catch (error) {
                console.log("Error running code:", error);
                set({
                    error: "Error running code",
                    executionResult: { code, output: "", error: "Error running code" },
                });
            } finally {
                set({ isRunning: false });
            }
        },
    };
});

export const getExecutionResult = () => useCodeEditorStore.getState().executionResult;
```

■ File: src/hooks/useSessionActivity.ts

Size: 5480 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■ ***Custom React Hook*** *- Reusable logic for React components*
■ *Contains: 4 imports, 1 exports, ~8 functions/constants*

■ *Uses Convex backend framework*

■ Functions & Logic Analysis:

Function: useSessionActivity
- *Parameters: { sessionId, userId, isActive = true }*
- *Performs database mutations (create/update/delete operations)*
- *Handles side effects and lifecycle events*
- *Uses refs for DOM access or mutable values*
- *Uses timers for delayed/repeated execution*
- *Includes error handling*
- *Returns data/object/JSX*

```
export function useSessionActivity({ sessionId, userId, isActive = true }: UseSessionActivityProps) {
  const updateActivity = useMutation(api.sessionActivity.updateUserActivity);
  const heartbeat = useMutation(api.collaboration.participantHeartbeat);

  const heartbeatIntervalRef = useRef<NodeJS.Timeout | null>(null);
  const lastActivityRef = useRef<number>(Date.now());

  // Send heartbeat to server
  const sendHeartbeat = useCallback(async () => {
    if (!sessionId || !userId || !isActiv...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
"use client";

import { useEffect, useRef, useCallback } from 'react';
import { useMutation } from 'convex/react';
import { api } from '../../convex/_generated/api';
import { Id } from '../../convex/_generated/dataModel';

interface UseSessionActivityProps {
  sessionId: Id<"collaborativeSessions"> | null;
  userId: string | null;
  isActive?: boolean;
}

export function useSessionActivity({ sessionId, userId, isActive = true }: UseSessionActivityProps) {
  const updateActivity = useMutation(api.sessionActivity.updateUserActivity);
  const heartbeat = useMutation(api.collaboration.participantHeartbeat);

  const heartbeatIntervalRef = useRef<NodeJS.Timeout | null>(null);
  const lastActivityRef = useRef<number>(Date.now());

  // Send heartbeat to server
  const sendHeartbeat = useCallback(async () => {
    if (!sessionId || !userId || !isActive) return;

    try {
      await heartbeat({ sessionId, userId });
    } catch (error) {
      console.error('Failed to send heartbeat:', error);
    }
  }, [sessionId, userId, isActive, heartbeat]);

  // Update user activity
  const updateUserActivity = useCallback(async () => {
    if (!sessionId || !userId || !isActive) return;

    const now = Date.now();
    // Throttle activity updates to once every 10 seconds
    if (now - lastActivityRef.current < 10000) return;

    lastActivityRef.current = now;
```

```
  try {
    await updateActivity({ sessionId, userId });
  } catch (error) {
    console.error('Failed to update activity:', error);
  }
}, [sessionId, userId, isActive, updateActivity]);

// Handle user activity events
const handleActivity = useCallback(() => {
  updateUserActivity();
}, [updateUserActivity]);

// Handle page unload and navigation
const handleBeforeUnload = useCallback(() => {
  if (sessionId && userId) {
    // Use sendBeacon for reliable departure notification
    const data = JSON.stringify({ sessionId, userId, action: 'leave' });

    // Try to send via navigator.sendBeacon (most reliable for page unload)
    if (navigator.sendBeacon) {
      const blob = new Blob([data], { type: 'application/json' });
      navigator.sendBeacon('/api/session-activity', blob);
    }
  }
}, [sessionId, userId]);

// Handle browser back/forward navigation
const handlePopState = useCallback(() => {
  // User navigated away from the collaboration page
  if (sessionId && userId) {
    const data = JSON.stringify({ sessionId, userId, action: 'leave' });

    if (navigator.sendBeacon) {
      const blob = new Blob([data], { type: 'application/json' });
      navigator.sendBeacon('/api/session-activity', blob);
    }
  }
}, [sessionId, userId]);

// Handle page visibility change (user switches tabs, minimizes window, etc.)
const handleVisibilityChange = useCallback(() => {
  if (document.visibilityState === 'visible') {
    // User is back, send heartbeat
    sendHeartbeat();
  } else {
    // User is away, we'll let the server-side timeout handle this
    // The session will be marked inactive after the threshold
  }
}, [sendHeartbeat]);

useEffect(() => {
  if (!sessionId || !userId || !isActive) {
    // Clear heartbeat if session becomes inactive
    if (heartbeatIntervalRef.current) {
      clearInterval(heartbeatIntervalRef.current);
      heartbeatIntervalRef.current = null;
    }
    return;
  }

  // Start heartbeat interval (every 30 seconds)
  heartbeatIntervalRef.current = setInterval(sendHeartbeat, 30000);

  // Activity event listeners
  const activityEvents = [
    'mousedown',
    'mousemove',
    'keypress',
    'scroll',
    'touchstart',
    'click'
  ];

  // Add activity listeners (throttled)
  let activityTimeout: NodeJS.Timeout | null = null;
  const throttledActivity = () => {
    if (activityTimeout) return;
    activityTimeout = setTimeout(() => {
      handleActivity();
```

```
        activityTimeout = null;
      }, 5000); // Throttle to once every 5 seconds
    };

    activityEvents.forEach(event => {
      window.addEventListener(event, throttledActivity, { passive: true });
    });

    // Page visibility change
    document.addEventListener('visibilitychange', handleVisibilityChange);

    // Page unload
    window.addEventListener('beforeunload', handleBeforeUnload);

    // Browser navigation (back/forward buttons)
    window.addEventListener('popstate', handlePopState);

    // Send initial heartbeat
    sendHeartbeat();

    // Cleanup
    return () => {
      if (heartbeatIntervalRef.current) {
        clearInterval(heartbeatIntervalRef.current);
      }

      if (activityTimeout) {
        clearTimeout(activityTimeout);
      }

      activityEvents.forEach(event => {
        window.removeEventListener(event, throttledActivity);
      });

      document.removeEventListener('visibilitychange', handleVisibilityChange);
      window.removeEventListener('beforeunload', handleBeforeUnload);
      window.removeEventListener('popstate', handlePopState);
    };
  }, [sessionId, userId, isActive, sendHeartbeat, handleActivity, handleVisibilityChange, handleBeforeUnload

  return {
    sendHeartbeat,
    updateUserActivity,
  };
}
```

■ File: src/utils/sessionId.ts

Size: 1563 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■ ***Utility Functions*** *- Helper functions and common utilities*
■ *Contains: 0 imports, 3 exports, ~3 functions/constants*

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
/**
 * Generate a unique session ID using timestamp, random values, and crypto
 * Format: [timestamp][random][crypto] -> ~15-20 characters
 * Example: "lmn4k2p7x3q9r5wx"
 */
export function generateSessionId(): string {
  // Base36 timestamp (more compact than base10)
  const timestamp = Date.now().toString(36);

  // Random component
  const random = Math.random().toString(36).substring(2, 8);
```

```
  // Crypto component for extra uniqueness
  const crypto = typeof window !== 'undefined' && window.crypto
    ? window.crypto.getRandomValues(new Uint8Array(3))
        .reduce((acc, byte) => acc + byte.toString(36), '')
    : Math.random().toString(36).substring(2, 5);

  return `${timestamp}${random}${crypto}`.toLowerCase();
}

/**
 * Validate if a session ID has the correct format
 */
export function isValidSessionId(sessionId: string): boolean {
  // Should be 15-25 characters, alphanumeric lowercase
  return /^[a-z0-9]{15,25}$/.test(sessionId);
}

/**
 * Extract timestamp from session ID (approximate creation time)
 */
export function getSessionCreationTime(sessionId: string): Date | null {
  try {
    // Extract first ~11 characters as timestamp
    const timestampStr = sessionId.substring(0, 11);
    const timestamp = parseInt(timestampStr, 36);

    // Validate the timestamp is reasonable (after 2020)
    if (timestamp > 1577836800000) { // Jan 1, 2020
      return new Date(timestamp);
    }

    return null;
  } catch {
    return null;
  }
}
```

■ File: src/app/(root)/_components/EditorPanel.tsx

Size: 6363 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■ *Contains: 10 imports, 1 exports, ~4 functions/constants*
■ *Integrates Clerk authentication*
■ *Integrates Monaco code editor*

■ Functions & Logic Analysis:

Function: EditorPanel
• Handles side effects and lifecycle events

```
function EditorPanel() {
    const clerk = useClerk();
    const { language, theme, fontSize, editor, setFontSize, setEditor } = useCodeEditorStore();
    const mounted = useMounted();
    useEffect(() => {
        const savedCode = localStorage.getItem(`editor-code-${language}`);
        const newCode = savedCode || LANGUAGE_CONFIG[language].defaultCode;
        if (editor) editor.setValue(newCode);
    useEffect(() => {
        const savedFontSize = localStorage.getItem("editor-font-size");
  ...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useCodeEditorStore } from "@/store/useCodeEditorStore";
import { useEffect } from "react";
import { defineMonacoThemes, LANGUAGE_CONFIG } from "../_constants";
import { Editor } from "@monaco-editor/react";
import Image from "next/image";
import { RotateCcwIcon, TypeIcon } from "lucide-react";
import { useClerk } from "@clerk/nextjs";
import { EditorPanelSkeleton } from "./EditorPanelSkeleton";
import useMounted from "@/hooks/useMounted";
import EditorRunButton from "./EditorRunButton";
function EditorPanel() {
    const clerk = useClerk();
    const { language, theme, fontSize, editor, setFontSize, setEditor } = useCodeEditorStore();
    const mounted = useMounted();
    useEffect(() => {
        const savedCode = localStorage.getItem(`editor-code-${language}`);
        const newCode = savedCode || LANGUAGE_CONFIG[language].defaultCode;
        if (editor) editor.setValue(newCode);
    useEffect(() => {
        const savedFontSize = localStorage.getItem("editor-font-size");
        if (savedFontSize) setFontSize(parseInt(savedFontSize));
    const handleRefresh = () => {
        const defaultCode = LANGUAGE_CONFIG[language].defaultCode;
        if (editor) editor.setValue(defaultCode);
    };
    const handleEditorChange = (value: string | undefined) => {
        if (value) localStorage.setItem(`editor-code-${language}`, value);
    };
    const handleFontSizeChange = (newSize: number) => {
        const size = Math.min(Math.max(newSize, 12), 24);
    };
    if (!mounted) return null;
    // [JSX RENDERING CODE EXCLUDED]
}
export default EditorPanel;
```

■ File: src/app/(root)/_components/OutputPanel.tsx

Size: 4312 bytes | Last Modified: 2025-10-31 16:53:49

> ■ ***File Overview:***
> ■ *Contains: 5 imports, 1 exports, ~2 functions/constants*

■ Functions & Logic Analysis:

> ***Function: OutputPanel***
> • *Manages component state*

```
function OutputPanel() {
    const { output, error, isRunning } = useCodeEditorStore();
    const [isCopied, setIsCopied] = useState(false);
    const hasContent = error || output;
    const handleCopy = async () => {
        if (!hasContent) return;
    };
    const mounted = useMounted();
    if (!mounted) return null;
    // [JSX RENDERING CODE EXCLUDED]
}
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
```

```
import { useCodeEditorStore } from "@/store/useCodeEditorStore";
import { AlertTriangle, CheckCircle, Clock, Copy, Terminal } from "lucide-react";
import { useState } from "react";
import RunningCodeSkeleton from "./RunningCodeSkeleton";
import useMounted from "@/hooks/useMounted";
function OutputPanel() {
    const { output, error, isRunning } = useCodeEditorStore();
    const [isCopied, setIsCopied] = useState(false);
    const hasContent = error || output;
    const handleCopy = async () => {
        if (!hasContent) return;
    };
    const mounted = useMounted();
    if (!mounted) return null;
    // [JSX RENDERING CODE EXCLUDED]
}
export default OutputPanel;
```

■ File: src/app/(root)/_components/ShareSnippetDialog.tsx

Size: 3424 bytes | Last Modified: 2025-10-31 16:53:49

> ■ ***File Overview:***
> ■ *Contains: 6 imports, 1 exports, ~2 functions/constants*
> ■ *Uses Convex backend framework*

■ Functions & Logic Analysis:

> ***Function: handleShare***
> *• Asynchronous function (uses async/await)*
> *• Parameters: e*
> *• Handles user events/interactions*

```
    const handleShare = async (e: React.FormEvent) => {
        try {
            const code = getCode();
        }
    };
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useCodeEditorStore } from "@/store/useCodeEditorStore";
import { useMutation } from "convex/react";
import { useState } from "react";
import { api } from "../../../../convex/_generated/api";
import { X } from "lucide-react";
import toast from "react-hot-toast";
function ShareSnippetDialog({ onClose }: { onClose: () => void }) {
    const [title, setTitle] = useState("");
    const [isSharing, setIsSharing] = useState(false);
    const { language, getCode } = useCodeEditorStore();
    const createSnippet = useMutation(api.snippets.createSnippet);
    const handleShare = async (e: React.FormEvent) => {
        try {
            const code = getCode();
        }
    };
    // [JSX RENDERING CODE EXCLUDED]
}
export default ShareSnippetDialog;
```

■ File: src/app/(root)/_components/LanguageSelector.tsx

Size: 8718 bytes | Last Modified: 2025-10-31 16:53:49

> ■ ***File Overview:***
> ■ *Contains: 7 imports, 1 exports, ~3 functions/constants*

■ Functions & Logic Analysis:

> ***Function: LanguageSelector***
> • *Parameters: { hasAccess }*
> • *Manages component state*
> • *Handles side effects and lifecycle events*
> • *Uses refs for DOM access or mutable values*

```
function LanguageSelector({ hasAccess }: { hasAccess: boolean }) {
    const [isOpen, setIsOpen] = useState(false);
    const mounted = useMounted();
    const { language, setLanguage } = useCodeEditorStore();
    const dropdownRef = useRef<HTMLDivElement>(null);
    const currentLanguageObj = LANGUAGE_CONFIG[language];
    useEffect(() => {
        const handleClickOutside = (event: MouseEvent) => {
            if (dropdownRef.current && !dropdownRef.current.contains(event.target as Node)) {
  ...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useCodeEditorStore } from "@/store/useCodeEditorStore";
import { useEffect, useRef, useState } from "react";
import { LANGUAGE_CONFIG } from "../_constants";
import { motion, AnimatePresence } from "framer-motion";
import Image from "next/image";
import { ChevronDownIcon, Lock, Sparkles } from "lucide-react";
import useMounted from "@/hooks/useMounted";
function LanguageSelector({ hasAccess }: { hasAccess: boolean }) {
    const [isOpen, setIsOpen] = useState(false);
    const mounted = useMounted();
    const { language, setLanguage } = useCodeEditorStore();
    const dropdownRef = useRef<HTMLDivElement>(null);
    const currentLanguageObj = LANGUAGE_CONFIG[language];
    useEffect(() => {
        const handleClickOutside = (event: MouseEvent) => {
            if (dropdownRef.current && !dropdownRef.current.contains(event.target as Node)) {
            }
        };
        // [JSX RENDERING CODE EXCLUDED]
    const handleLanguageSelect = (langId: string) => {
        if (!hasAccess && langId !== "javascript") return;
    };
    if (!mounted) return null;
    // [JSX RENDERING CODE EXCLUDED]
                                // [JSX RENDERING CODE EXCLUDED]
}
export default LanguageSelector;
```

■ File: src/app/(root)/_components/ThemeSelector.tsx

Size: 6428 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■ *Contains: 6 imports, 1 exports, ~3 functions/constants*

■ Functions & Logic Analysis:

Function: ThemeSelector
• Manages component state
• Handles side effects and lifecycle events
• Uses refs for DOM access or mutable values

```
function ThemeSelector() {
    const [isOpen, setIsOpen] = useState(false);
    const mounted = useMounted();
    const { theme, setTheme } = useCodeEditorStore();
    const dropdownRef = useRef<HTMLDivElement>(null);
    const currentTheme = THEMES.find((t) => t.id === theme);
    useEffect(() => {
        const handleClickOutside = (event: MouseEvent) => {
            if (dropdownRef.current && !dropdownRef.current.contains(event.target as Node)) {
            }
        };
        // [JSX REND...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useCodeEditorStore } from "@/store/useCodeEditorStore";
import React, { useEffect, useRef, useState } from "react";
import { THEMES } from "../_constants";
import { AnimatePresence, motion } from "framer-motion";
import { CircleOff, Cloud, Github, Laptop, Moon, Palette, Sun } from "lucide-react";
import useMounted from "@/hooks/useMounted";
const THEME_ICONS: Record<string, React.ReactNode> = {
};
function ThemeSelector() {
    const [isOpen, setIsOpen] = useState(false);
    const mounted = useMounted();
    const { theme, setTheme } = useCodeEditorStore();
    const dropdownRef = useRef<HTMLDivElement>(null);
    const currentTheme = THEMES.find((t) => t.id === theme);
    useEffect(() => {
        const handleClickOutside = (event: MouseEvent) => {
            if (dropdownRef.current && !dropdownRef.current.contains(event.target as Node)) {
            }
        };
        // [JSX RENDERING CODE EXCLUDED]
    if (!mounted) return null;
    // [JSX RENDERING CODE EXCLUDED]
}
export default ThemeSelector;
```

■ File: src/components/MultiFileEditor.tsx

Size: 16069 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■ *Contains: 9 imports, 1 exports, ~10 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*

■ *Integrates Monaco code editor*

■ Functions & Logic Analysis:

Function: MultiFileEditor
- *Parameters: { selectedFileId, onFileClose }*
- *Performs database mutations (create/update/delete operations)*
- *Fetches data from the database*
- *Makes HTTP/API requests*
- *Manages component state*
- *Handles side effects and lifecycle events*
- *Uses timers for delayed/repeated execution*
- *Transforms/processes data arrays*
- *Performs validation checks*

```
export default function MultiFileEditor({ selectedFileId, onFileClose }: MultiFileEditorProps) {
  const { user } = useUser();
  const mounted = useMounted();
  const [tabs, setTabs] = useState<EditorTab[]>([]);
  const [activeTabId, setActiveTabId] = useState<Id<"practiceFiles"> | null>(null);
  const [unsavedChanges, setUnsavedChanges] = useState<Map<string, string>>(new Map());
  const [isRunning, setIsRunning] = useState(false);
  const [output, setOutput] = useState("");
  const [error, set...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState, useEffect, useCallback } from "react";
import { useQuery, useMutation } from "convex/react";
import { api } from "../../convex/_generated/api";
import { useUser } from "@clerk/nextjs";
import { Editor } from "@monaco-editor/react";
import { defineMonacoThemes, LANGUAGE_CONFIG } from "@/app/(root)/_constants";
import { XIcon, FileIcon, PlayIcon, Loader2Icon } from "lucide-react";
import { Id } from "../../convex/_generated/dataModel";
import useMounted from "@/hooks/useMounted";
interface EditorTab {
}
interface MultiFileEditorProps {
}
export default function MultiFileEditor({ selectedFileId, onFileClose }: MultiFileEditorProps) {
  const { user } = useUser();
  const mounted = useMounted();
  const [tabs, setTabs] = useState<EditorTab[]>([]);
  const [activeTabId, setActiveTabId] = useState<Id<"practiceFiles"> | null>(null);
  const [unsavedChanges, setUnsavedChanges] = useState<Map<string, string>>(new Map());
  const [isRunning, setIsRunning] = useState(false);
  const [output, setOutput] = useState("");
  const [error, setError] = useState<string | null>(null);
  const [showOutput, setShowOutput] = useState(false);
  // Mutations
  const updateFile = useMutation(api.files.updateFile);
  // Queries
  const activeFile = useQuery(
  const openFile = useCallback((fileId: Id<"practiceFiles">) => {
    // Check if file is already open
    const existingTab = tabs.find(tab => tab.fileId === fileId);
    if (existingTab) {
    }
    // Optimistic update: Create tab immediately with loading state
    const optimisticTab: EditorTab = {
    };
  // Handle file selection from file tree
  useEffect(() => {
    if (selectedFileId && user?.id) {
    }
  // Override Ctrl+S to save file instead of downloading webpage
```

```
useEffect(() => {
  const handleKeyDown = async (event: KeyboardEvent) => {
    if (event.ctrlKey && event.key === 's') {
      if (activeTabId && user?.id) {
        const editorContent = unsavedChanges.get(activeTabId);
        if (editorContent !== undefined) {
          try {
            // Remove from unsaved changes
              const newMap = new Map(prev);
            // Update tab dirty state
              prevTabs.map(tab =>
          }
        }
      }
    }
  };
  // Add global event listener
  // Cleanup
  // [JSX RENDERING CODE EXCLUDED]
  };
// Update tabs when activeFile data is loaded
useEffect(() => {
  if (activeFile && activeTabId) {
    const existingTab = tabs.find(tab => tab.fileId === activeTabId);
    if (!existingTab) {
      // This shouldn't happen with optimistic updates, but keep as fallback
      const newTab: EditorTab = {
      };
      // Update the optimistic tab with real data
        prevTabs.map(tab =>
              }
    }
  }
const closeTab = (fileId: Id<"practiceFiles">, event?: React.MouseEvent) => {
  if (event) {
  }
  // Check for unsaved changes
  const hasUnsavedChanges = unsavedChanges.has(fileId);
  if (hasUnsavedChanges) {
    const shouldClose = confirm("You have unsaved changes. Close anyway?");
    if (!shouldClose) return;
  }
  // Get remaining tabs before removing current one
  const remainingTabs = tabs.filter(tab => tab.fileId !== fileId);
  // Remove tab
  // Remove unsaved changes
    const newMap = new Map(prev);
  // Update active tab
  if (activeTabId === fileId) {
    if (remainingTabs.length > 0) {
      // Switch to the last remaining tab
    }
  }
};
const handleEditorChange = (value: string | undefined) => {
  if (value === undefined || !activeTabId) return;
  // Store unsaved changes
  const newUnsavedChanges = new Map(unsavedChanges);
  // Mark tab as dirty
    prevTabs.map(tab =>
};
// Auto-save effect
useEffect(() => {
  if (!activeTabId || !user?.id) return;
  const interval = setInterval(async () => {
    const editorContent = unsavedChanges.get(activeTabId);
    if (editorContent !== undefined) {
      try {
        // Remove from unsaved changes
        const newUnsavedChanges = new Map(unsavedChanges);
        // Update tab dirty state
          prevTabs.map(tab =>
      }
    }
  // [JSX RENDERING CODE EXCLUDED]
// Run code function
const runCode = async () => {
  if (!activeFile || !activeTabId) return;
  const code = unsavedChanges.get(activeTabId) ?? activeFile.code;
```

```
    if (!code.trim()) {
    }
    try {
      const runtime = LANGUAGE_CONFIG[activeFile.language]?.pistonRuntime;
      if (!runtime) {
      }
      const response = await fetch("https://emkc.org/api/v2/piston/execute", {
        },
      const data = await response.json();
      // Handle API-level errors
      if (data.message) {
      }
      // Handle compilation errors
      if (data.compile && data.compile.code !== 0) {
        const error = data.compile.stderr || data.compile.output;
      }
      // Handle runtime errors
      if (data.run && data.run.code !== 0) {
        const error = data.run.stderr || data.run.output;
      }
      // Success
      const output = data.run.output;
    }
  };
  if (!mounted) return null;
  // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/MultiFileEditorSimple.tsx

Size: 6199 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■ *Contains: 9 imports, 1 exports, ~6 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*
■ *Integrates Monaco code editor*

■ Functions & Logic Analysis:

Function: MultiFileEditor
• Parameters: { selectedFileId }
• Performs database mutations (create/update/delete operations)
• Fetches data from the database
• Manages component state
• Handles side effects and lifecycle events
• Uses timers for delayed/repeated execution
• Transforms/processes data arrays
• Performs validation checks

```
export default function MultiFileEditor({ selectedFileId }: MultiFileEditorProps) {
  const { user } = useUser();
  const mounted = useMounted();
  const [tabs, setTabs] = useState<EditorTab[]>([]);
  const [activeTabId, setActiveTabId] = useState<Id<"practiceFiles"> | null>(null);
  const [unsavedChanges, setUnsavedChanges] = useState<Map<string, string>>(new Map());
  // Mutations
  const updateFile = useMutation(api.files.updateFile);
  // Queries
  const activeFile = useQuery(
  const openFi...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState, useEffect, useCallback } from "react";
import { useQuery, useMutation } from "convex/react";
import { api } from "../../convex/_generated/api";
import { useUser } from "@clerk/nextjs";
import { Editor } from "@monaco-editor/react";
import { defineMonacoThemes, LANGUAGE_CONFIG } from "@/app/(root)/_constants";
import { FileIcon } from "lucide-react";
import { Id } from "../../convex/_generated/dataModel";
import useMounted from "@/hooks/useMounted";
interface EditorTab {
}
interface MultiFileEditorProps {
}
export default function MultiFileEditor({ selectedFileId }: MultiFileEditorProps) {
  const { user } = useUser();
  const mounted = useMounted();
  const [tabs, setTabs] = useState<EditorTab[]>([]);
  const [activeTabId, setActiveTabId] = useState<Id<"practiceFiles"> | null>(null);
  const [unsavedChanges, setUnsavedChanges] = useState<Map<string, string>>(new Map());
  // Mutations
  const updateFile = useMutation(api.files.updateFile);
  // Queries
  const activeFile = useQuery(
  const openFile = useCallback((fileId: Id<"practiceFiles">) => {
    // Check if file is already open
    const existingTab = tabs.find(tab => tab.fileId === fileId);
    if (existingTab) {
    }
  // Handle file selection from file tree
  useEffect(() => {
    if (selectedFileId && user?.id) {
    }
  // Update tabs when activeFile data is loaded
  useEffect(() => {
    if (activeFile && activeTabId) {
      const existingTab = tabs.find(tab => tab.fileId === activeTabId);
      if (!existingTab) {
        const newTab: EditorTab = {
        };
      }
    }
  const handleEditorChange = (value: string | undefined) => {
    if (value === undefined || !activeTabId) return;
    // Store unsaved changes
    const newUnsavedChanges = new Map(unsavedChanges);
    // Mark tab as dirty
      prevTabs.map(tab =>
  };
  // Auto-save effect
  useEffect(() => {
    if (!activeTabId || !user?.id) return;
    const interval = setInterval(async () => {
      const editorContent = unsavedChanges.get(activeTabId);
      if (editorContent !== undefined) {
        try {
          // Remove from unsaved changes
          const newUnsavedChanges = new Map(unsavedChanges);
          // Update tab dirty state
            prevTabs.map(tab =>
        }
      }
    // [JSX RENDERING CODE EXCLUDED]
  if (!mounted) return null;
  // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/FileOperationsPanel.tsx

Size: 15164 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■ *Contains: 7 imports, 1 exports, ~8 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*

■ Functions & Logic Analysis:

Function: handleKeyDown
• Parameters: event
• Handles user events/interactions

```
  const handleKeyDown = (event: KeyboardEvent) => {
    if (event.key === 'Escape') {
      if (showMoveDialog) {
      }
    }
  };
```

Function: handleDeleteSelected
• Asynchronous function (uses async/await)
• Handles user events/interactions

```
const handleDeleteSelected = async () => {
  if (!user?.id || selectedFiles.length === 0) return;
  const confirmDelete = confirm(
  if (!confirmDelete) return;
  try {
    for (const fileId of selectedFiles) {
    }
  }
};
```

Function: handleMoveSelected
• Asynchronous function (uses async/await)
• Handles user events/interactions

```
const handleMoveSelected = async () => {
  if (!user?.id || selectedFiles.length === 0) return;
  try {
    for (const fileId of selectedFiles) {
    }
  }
};
```

Function: handleExportSelected
• Transforms/processes data arrays
• Handles user events/interactions

```
const handleExportSelected = () => {
  if (!allFiles || selectedFiles.length === 0) return;
  const filesToExport = allFiles.filter(file =>
  const exportData = {
    files: filesToExport.map(file => ({
  };
  const blob = new Blob([JSON.stringify(exportData, null, 2)], {
  const url = URL.createObjectURL(blob);
  const a = document.createElement('a');
};
const filteredAndSortedFiles = allFiles
  ?.filter(file => {
    if (filterType === "all") return true;
    let c...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState, useEffect } from "react";
import { useQuery, useMutation } from "convex/react";
import { api } from "../../convex/_generated/api";
import { useUser } from "@clerk/nextjs";
import toast from "react-hot-toast";
import {
import { Id } from "../../convex/_generated/dataModel";
interface FileOperationsPanelProps {
}
type SortOption = "name" | "date" | "size" | "type";
type SortOrder = "asc" | "desc";
export default function FileOperationsPanel({
  const { user } = useUser();
  const [sortBy, setSortBy] = useState<SortOption>("date");
  const [sortOrder, setSortOrder] = useState<SortOrder>("desc");
  const [filterType, setFilterType] = useState<string>("all");
  const [showMoveDialog, setShowMoveDialog] = useState(false);
  const [targetFolderId, setTargetFolderId] = useState<Id<"folders"> | undefined>();
  // Handle escape key
  useEffect(() => {
    const handleKeyDown = (event: KeyboardEvent) => {
      if (event.key === 'Escape') {
        if (showMoveDialog) {
        }
      }
    };
    // [JSX RENDERING CODE EXCLUDED]
  // Mutations
  const deleteFile = useMutation(api.files.deleteFile);
  const moveFile = useMutation(api.files.moveFile);
  // Queries
  const allFiles = useQuery(
  const folderTree = useQuery(
  const handleDeleteSelected = async () => {
    if (!user?.id || selectedFiles.length === 0) return;
    const confirmDelete = confirm(
    if (!confirmDelete) return;
    try {
      for (const fileId of selectedFiles) {
      }
    }
  };
  const handleMoveSelected = async () => {
    if (!user?.id || selectedFiles.length === 0) return;
    try {
      for (const fileId of selectedFiles) {
      }
    }
  };
  const handleExportSelected = () => {
    if (!allFiles || selectedFiles.length === 0) return;
    const filesToExport = allFiles.filter(file =>
    const exportData = {
      files: filesToExport.map(file => ({
    };
    const blob = new Blob([JSON.stringify(exportData, null, 2)], {
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
  };
  const filteredAndSortedFiles = allFiles
    ?.filter(file => {
      if (filterType === "all") return true;
      let comparison = 0;
      switch (sortBy) {
        case "name":
        case "date":
        case "type":
        case "size":
      }
  const uniqueLanguages = Array.from(
    new Set(allFiles?.map(file => file.language) || [])
  type FolderTreeItem = {
  };
  const renderFolderOption = (folder: FolderTreeItem, depth = 0) => (
  const renderFolderTree = (folders: FolderTreeItem[], depth = 0): JSX.Element[] => {
  };
```

```
  // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/FileTree.tsx

Size: 22548 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■ *Contains: 7 imports, 1 exports, ~14 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*

■ Functions & Logic Analysis:

Function: FileTree
• Parameters: { onFileSelect, selectedFileId, onSearchOpen, onOperationsOpen }
• Performs database mutations (create/update/delete operations)
• Fetches data from the database
• Manages component state
• Performs validation checks

```
export default function FileTree({ onFileSelect, selectedFileId, onSearchOpen, onOperationsOpen }: FileTreeP
    const { user } = useUser();
    const [expandedFolders, setExpandedFolders] = useState<Set<string>>(new Set());
    const [contextMenu, setContextMenu] = useState<{
    // State for inline creation
    const [creatingItem, setCreatingItem] = useState<{
    // State for handling duplicate name errors
    const [duplicateError, setDuplicateError] = useState<string | null>(null);
...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState } from "react";
import { useQuery, useMutation } from "convex/react";
import { api } from "../../convex/_generated/api";
import { useUser } from "@clerk/nextjs";
import {
import { Id } from "../../convex/_generated/dataModel";
import InlineInput from "./InlineInput";
interface FileTreeProps {
}
export default function FileTree({ onFileSelect, selectedFileId, onSearchOpen, onOperationsOpen }: FileTreeP
    const { user } = useUser();
    const [expandedFolders, setExpandedFolders] = useState<Set<string>>(new Set());
    const [contextMenu, setContextMenu] = useState<{
    // State for inline creation
    const [creatingItem, setCreatingItem] = useState<{
    // State for handling duplicate name errors
    const [duplicateError, setDuplicateError] = useState<string | null>(null);
    // Mutations
    const createFolder = useMutation(api.folders.createFolder);
    const createFile = useMutation(api.files.createFile);
    const deleteFolder = useMutation(api.folders.deleteFolder);
    const deleteFile = useMutation(api.files.deleteFile);
    // Queries
    const folderTree = useQuery(
    const toggleFolder = (folderId: string) => {
        const newExpanded = new Set(expandedFolders);
        if (newExpanded.has(folderId)) {
        }
```

```
};
const handleContextMenu = (e: React.MouseEvent, type: 'folder' | 'file' | 'root', id?: string) => {
};
const handleCreateFolder = async (parentId?: Id<"folders">) => {
    if (!user?.id) return;
    // Auto-expand the parent folder if creating inside it
    if (parentId && !expandedFolders.has(parentId)) {
    }
};
const handleCreateFile = async (folderId?: Id<"folders">) => {
    if (!user?.id) return;
    // Auto-expand the target folder if creating inside it
    if (folderId && !expandedFolders.has(folderId)) {
    }
};
const handleCreateSave = async (name: string) => {
    if (!user?.id || !creatingItem) return;
    // Client-side duplicate validation BEFORE making API call
    if (creatingItem.type === 'folder') {
        // Check for existing folders in the same parent
        // eslint-disable-next-line @typescript-eslint/no-explicit-any
        const getAllFolders = (folders: any[]): any[] => {
            // eslint-disable-next-line @typescript-eslint/no-explicit-any
            let allFolders: any[] = [];
                if (folder.children) {
                }
        };
        if (folderTree?.folders) {
            const allFolders = getAllFolders(folderTree.folders);
            const existingFolder = allFolders.find(folder =>
            if (existingFolder) {
            }
        }
        // Check for existing files in the same folder
        // eslint-disable-next-line @typescript-eslint/no-explicit-any
        const getAllFiles = (folders: any[]): any[] => {
            // eslint-disable-next-line @typescript-eslint/no-explicit-any
            let allFiles: any[] = [];
                if (folder.files) {
                }
                if (folder.children) {
                }
        };
        if (folderTree?.folders || folderTree?.files) {
            let allFiles = folderTree.files || [];
            if (folderTree.folders) {
            }
            const existingFile = allFiles.find(file =>
            if (existingFile) {
            }
        }
    }
    try {
        if (creatingItem.type === 'folder') {
            // Extract language from extension
            const extension = name.split('.').pop()?.toLowerCase() || 'txt';
            const languageMap: Record<string, string> = {
            };
            const language = languageMap[extension] || 'javascript';
            const fileId = await createFile({
        }
        // Handle any backend errors (this should rarely happen with client-side validation)
        if (error instanceof Error && error.message.includes('already exists')) {
        }
        // For other errors, cancel the creation
    }
};
const handleCreateCancel = () => {
};
const handleDelete = async (type: 'folder' | 'file', id: string) => {
    if (!user?.id) return;
    const confirmMessage = type === 'folder'
    if (!confirm(confirmMessage)) return;
    try {
        if (type === 'folder') {
        }
        // You can add a toast notification here instead of alert
    }
};
```

```
    // eslint-disable-next-line @typescript-eslint/no-explicit-any
    const renderFolder = (folder: any, depth = 0) => {
        const isExpanded = expandedFolders.has(folder._id);
        const paddingClass = depth === 0 ? 'pl-2' : depth === 1 ? 'pl-6' : depth === 2 ? 'pl-10' : 'pl-14';
        // [JSX RENDERING CODE EXCLUDED]
    };
    // eslint-disable-next-line @typescript-eslint/no-explicit-any
    const renderFile = (file: any, depth = 0) => {
        const isSelected = selectedFileId === file._id;
        const paddingClass = depth === 0 ? 'pl-2' : depth === 1 ? 'pl-6' : depth === 2 ? 'pl-10' : 'pl-14';
        // [JSX RENDERING CODE EXCLUDED]
    };
    if (!folderTree) {
        // [JSX RENDERING CODE EXCLUDED]
    }
    // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/collaboration/SessionManager.tsx

Size: 32310 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■■ ***Business Logic*** *- Core application functionality*
■ *Contains: 8 imports, 1 exports, ~10 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*

■ Functions & Logic Analysis:

Function: SessionManager
• *Parameters: { onSessionSelect }*
• *Performs database mutations (create/update/delete operations)*
• *Fetches data from the database*
• *Manages component state*
• *Handles side effects and lifecycle events*
• *Transforms/processes data arrays*
• *Performs validation checks*

```
export default function SessionManager({ onSessionSelect }: SessionManagerProps) {
  const { user } = useUser();
  const [showCreateForm, setShowCreateForm] = useState(false);
  const [activeTab, setActiveTab] = useState<'my' | 'public' | 'savedPublic'>('my');
  const [isClient, setIsClient] = useState(false);
  // Modal states
  const [showDeleteConfirm, setShowDeleteConfirm] = useState<string | null>(null);
  const [showSaveModal, setShowSaveModal] = useState<{ sessionId: string; sessionName: ...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState, useEffect } from 'react';
import { useQuery, useMutation } from 'convex/react';
import { useUser } from '@clerk/nextjs';
import { api } from '../../../convex/_generated/api';
import { Id } from '../../../convex/_generated/dataModel';
import { toast } from 'react-toastify';
import {
import { ConfirmModal, InputModal } from '../ui/Modal';
interface SessionManagerProps {
}
```

```
export default function SessionManager({ onSessionSelect }: SessionManagerProps) {
  const { user } = useUser();
  const [showCreateForm, setShowCreateForm] = useState(false);
  const [activeTab, setActiveTab] = useState<'my' | 'public' | 'savedPublic'>('my');
  const [isClient, setIsClient] = useState(false);
  // Modal states
  const [showDeleteConfirm, setShowDeleteConfirm] = useState<string | null>(null);
  const [showSaveModal, setShowSaveModal] = useState<{ sessionId: string; sessionName: string } | null>(nul
  const [showEditModal, setShowEditModal] = useState<{
  useEffect(() => {
  // Queries
  const userSessions = useQuery(
  const publicSessions = useQuery(
  // Get public saved sessions
  const publicSavedSessions = useQuery(
  // Get participant counts for real-time updates
  const userSessionIds = userSessions?.map(s => s._id).filter(Boolean) || [];
  const publicSessionIds = publicSessions?.map(s => s._id).filter(Boolean) || [];
  const allSessionIds = [...userSessionIds, ...publicSessionIds];
  const participantCounts = useQuery(
  // Create a map for quick lookup of participant counts
  const participantCountMap = participantCounts?.reduce((acc, item) => {
  // Mutations
  const createSession = useMutation(api.collaboration.createSession);
  const joinSession = useMutation(api.collaboration.joinSession);
  const updateSession = useMutation(api.collaboration.updateSession);
  const deleteSession = useMutation(api.collaboration.deleteSession);
  const saveSessionToCollection = useMutation(api.collaboration.saveSessionToCollection);
  // New query for session limit validation
  const sessionLimitCheck = useQuery(
  // Check which sessions are already saved
  const sessionSaveStatus = useQuery(
  // Get saved session info
  const savedSessionInfo = useQuery(
  // Create a map for quick lookup
  const saveStatusMap = sessionSaveStatus?.reduce((acc, item) => {
  const handleCreateSession = async (formData: FormData) => {
    if (!user?.id) return;
    // Check session limit
    if (sessionLimitCheck && !sessionLimitCheck.canCreate) {
    }
    const name = formData.get('name') as string;
    const description = formData.get('description') as string;
    const language = formData.get('language') as string;
    const isPublic = formData.get('isPublic') === 'on';
    const maxUsers = parseInt(formData.get('maxUsers') as string) || 5;
    try {
      const result = await createSession({
      // Navigate to the session using sessionKey
      const errorMessage = error instanceof Error ? error.message : 'Failed to create session. Please try ag
    }
  };
  const handleJoinSession = async (session: { _id: Id<"collaborativeSessions">; sessionKey: string }) => {
    if (!user?.id) return;
    try {
    }
  };
  const handleUpdateSession = async (updateData: {
    };
    if (!user?.id) return;
    try {
      const { sessionId, ...restData } = updateData;
    }
  };
  const confirmDeleteSession = async (sessionId: string) => {
    if (!user?.id) return;
    try {
    }
  };
  const handleSaveSession = async (sessionId: string, sessionName: string, name: string, description?: strin
    if (!user?.id) return;
    try {
      const result = await saveSessionToCollection({
      const errorMessage = error instanceof Error ? error.message : 'Failed to save session. Please try agai
      if (errorMessage.includes('DUPLICATE_SAVE')) {
      }
    }
  };
  const formatTimeAgo = (timestamp: number) => {
```

```
    if (!isClient) return 'Recently'; // Fallback for SSR
    const diff = Date.now() - timestamp;
    const minutes = Math.floor(diff / 60000);
    const hours = Math.floor(diff / 3600000);
    const days = Math.floor(diff / 86400000);
    if (days > 0) return `${days}d ago`;
    if (hours > 0) return `${hours}h ago`;
    if (minutes > 0) return `${minutes}m ago`;
  };
  // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/collaboration/CollaborationIntegration.tsx

Size: 6292 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■■ ***Business Logic*** *- Core application functionality*
■ *Contains: 7 imports, 1 exports, ~2 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*

■ Functions & Logic Analysis:

Function: handleCreateSession
• *Asynchronous function (uses async/await)*
• *Parameters: isPublic*
• *Handles user events/interactions*

```
  const handleCreateSession = async (isPublic: boolean) => {
    if (!user?.id) return;
    try {
      const result = await createSession({
      // Redirect to collaboration page with the new session using sessionKey
    }
  };
  // [JSX RENDERING CODE EXCLUDED]
}
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState } from 'react';
import { useUser } from '@clerk/nextjs';
import { useMutation } from 'convex/react';
import { api } from '../../../convex/_generated/api';
import { Users, UserPlus } from 'lucide-react';
import Link from 'next/link';
import toast from 'react-hot-toast';
interface CollaborationIntegrationProps {
}
export default function CollaborationIntegration({
  const { user } = useUser();
  const [isCreating, setIsCreating] = useState(false);
  const [showModal, setShowModal] = useState(false);
  const createSession = useMutation(api.collaboration.createSession);
  const handleCreateSession = async (isPublic: boolean) => {
    if (!user?.id) return;
    try {
      const result = await createSession({
      // Redirect to collaboration page with the new session using sessionKey
    }
  };
```

```
  // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/collaboration/CollaborativeEditor.tsx

Size: 13529 bytes | Last Modified: 2025-10-31 16:53:49

> ■ ***File Overview:***
> ■■ ***Business Logic*** *- Core application functionality*
> ■ *Contains: 10 imports, 1 exports, ~7 functions/constants*
> ■ *Uses Convex backend framework*
> ■ *Integrates Clerk authentication*

■ Functions & Logic Analysis:

> ***Function: CollaborativeEditor***
> *• Parameters: { sessionId, onLeaveSession }*
> *• Performs database mutations (create/update/delete operations)*
> *• Fetches data from the database*
> *• Manages component state*
> *• Handles side effects and lifecycle events*
> *• Performs validation checks*

```
export default function CollaborativeEditor({ sessionId, onLeaveSession }: CollaborativeEditorProps) {
  const { user } = useUser();
  const [showChat, setShowChat] = useState(false);
  const [showParticipants, setShowParticipants] = useState(false);
  const [chatMessage, setChatMessage] = useState('');
  const [showLeaveConfirm, setShowLeaveConfirm] = useState(false);
  const [isClient, setIsClient] = useState(false);
  // File system states
  const [showFileExplorer, setShowFileExplorer] = use...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState, useEffect } from 'react';
import { useQuery, useMutation } from 'convex/react';
import { useUser } from '@clerk/nextjs';
import { api } from '../../../convex/_generated/api';
import { Id } from '../../../convex/_generated/dataModel';
import {
import { ConfirmModal } from '../ui/Modal';
import CollaborativeFileTree from './CollaborativeFileTree';
import MultiSessionFileEditor from './MultiSessionFileEditor';
import '../../styles/vscode-scrollbar.css';
interface CollaborativeEditorProps {
}
export default function CollaborativeEditor({ sessionId, onLeaveSession }: CollaborativeEditorProps) {
  const { user } = useUser();
  const [showChat, setShowChat] = useState(false);
  const [showParticipants, setShowParticipants] = useState(false);
  const [chatMessage, setChatMessage] = useState('');
  const [showLeaveConfirm, setShowLeaveConfirm] = useState(false);
  const [isClient, setIsClient] = useState(false);
  // File system states
  const [showFileExplorer, setShowFileExplorer] = useState(true);
  const [selectedFileId, setSelectedFileId] = useState<Id<"sessionFiles"> | undefined>();
  useEffect(() => {
  // Keyboard shortcut handler for Ctrl + B to toggle file explorer
  useEffect(() => {
    const handleKeyDown = (event: KeyboardEvent) => {
```

```
      // Check for Ctrl + B to toggle file explorer
      if (event.ctrlKey && event.key === 'b') {
      }
    };
    // Add event listener
    // Cleanup function
    // [JSX RENDERING CODE EXCLUDED]
    };
  // Language configurations for code execution
  // Moved to MultiSessionFileEditor where it's actually used
  // Queries
  const session = useQuery(
  const chatMessages = useQuery(
  // Mutations
  const leaveSession = useMutation(api.collaboration.leaveSession);
  const sendChatMessage = useMutation(api.collaboration.sendChatMessage);
  // Handle leaving session
  const handleLeaveSession = async () => {
    if (!user?.id) return;
  };
  const confirmLeaveSession = async () => {
    if (!user?.id) return;
    try {
    }
  };
  // Handle sending chat message
  const handleSendMessage = async () => {
    if (!user?.id || !chatMessage.trim()) return;
    try {
    }
  };
  const currentUserPermission = session?.participants?.find(p => p.userId === user?.id)?.permission || 'read
  const canEdit = currentUserPermission === 'write';
  if (!session) {
    // [JSX RENDERING CODE EXCLUDED]
  }
  // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/collaboration/CollaborativeFileTree.tsx

Size: 22586 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■■ ***Business Logic*** *- Core application functionality*
■ *Contains: 7 imports, 1 exports, ~14 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*

■ Functions & Logic Analysis:

Function: toggleFolder
• Parameters: folderId

```
    const toggleFolder = (folderId: string) => {
        const newExpanded = new Set(expandedFolders);
        if (newExpanded.has(folderId)) {
        }
    };
```

Function: handleContextMenu
• Parameters: e, type, id?
• Handles user events/interactions

```
    const handleContextMenu = (e: React.MouseEvent, type: 'folder' | 'file' | 'root', id?: string) => {
    };
```

Function: handleCreateFolder
• Asynchronous function (uses async/await)
• Parameters: parentId?
• Handles user events/interactions

```
    const handleCreateFolder = async (parentId?: Id<"sessionFolders">) => {
        if (!user?.id) return;
        // Auto-expand the parent folder if creating inside it
        if (parentId && !expandedFolders.has(parentId)) {
        }
    };
```

Function: handleCreateFile
• Asynchronous function (uses async/await)
• Parameters: folderId?
• Handles user events/interactions

```
    const handleCreateFile = async (folderId?: Id<"sessionFolders">) => {
        if (!user?.id) return;
        // Auto-expand the target folder if creating inside it
        if (folderId && !expandedFolders.has(folderId)) {
        }
    };
```

Function: handleCreateSave
• Asynchronous function (uses async/await)
• Parameters: name
• Performs validation checks
• Handles user events/interactions

```
    const handleCreateSave = async (name: string) => {
        if (!user?.id || !creatingItem) return;
        // Client-side duplicate validation
        if (creatingItem.type === 'folder') {
            // Check for existing folders in the same parent
            // eslint-disable-next-line @typescript-eslint/no-explicit-any
            const getAllFolders = (folders: any[]): any[] => {
                // eslint-disable-next-line @typescript-eslint/no-explicit-any
                let allFolder...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState } from "react";
import { useQuery, useMutation } from "convex/react";
import { api } from "../../../convex/_generated/api";
import { useUser } from "@clerk/nextjs";
import {
import { Id } from "../../../convex/_generated/dataModel";
import InlineInput from "../InlineInput";
interface CollaborativeFileTreeProps {
}
export default function CollaborativeFileTree({
    const { user } = useUser();
    const [expandedFolders, setExpandedFolders] = useState<Set<string>>(new Set());
    const [contextMenu, setContextMenu] = useState<{
    // State for inline creation
    const [creatingItem, setCreatingItem] = useState<{
    // State for handling duplicate name errors
    const [duplicateError, setDuplicateError] = useState<string | null>(null);
    // Mutations
    const createFolder = useMutation(api.sessionFolders.createFolder);
    const createFile = useMutation(api.sessionFiles.createFile);
    const deleteFolder = useMutation(api.sessionFolders.deleteFolder);
```

```
const deleteFile = useMutation(api.sessionFiles.deleteFile);
// Queries
const folderTree = useQuery(
const toggleFolder = (folderId: string) => {
    const newExpanded = new Set(expandedFolders);
    if (newExpanded.has(folderId)) {
    }
};
const handleContextMenu = (e: React.MouseEvent, type: 'folder' | 'file' | 'root', id?: string) => {
};
const handleCreateFolder = async (parentId?: Id<"sessionFolders">) => {
    if (!user?.id) return;
    // Auto-expand the parent folder if creating inside it
    if (parentId && !expandedFolders.has(parentId)) {
    }
};
const handleCreateFile = async (folderId?: Id<"sessionFolders">) => {
    if (!user?.id) return;
    // Auto-expand the target folder if creating inside it
    if (folderId && !expandedFolders.has(folderId)) {
    }
};
const handleCreateSave = async (name: string) => {
    if (!user?.id || !creatingItem) return;
    // Client-side duplicate validation
    if (creatingItem.type === 'folder') {
        // Check for existing folders in the same parent
        // eslint-disable-next-line @typescript-eslint/no-explicit-any
        const getAllFolders = (folders: any[]): any[] => {
            // eslint-disable-next-line @typescript-eslint/no-explicit-any
            let allFolders: any[] = [];
                if (folder.children) {
                }
        };
        if (folderTree?.folders) {
            const allFolders = getAllFolders(folderTree.folders);
            const existingFolder = allFolders.find(folder =>
            if (existingFolder) {
            }
        }
        // Check for existing files in the same folder
        // eslint-disable-next-line @typescript-eslint/no-explicit-any
        const getAllFiles = (folders: any[]): any[] => {
            // eslint-disable-next-line @typescript-eslint/no-explicit-any
            let allFiles: any[] = [];
                if (folder.files) {
                }
                if (folder.children) {
                }
        };
        if (folderTree?.folders || folderTree?.files) {
            let allFiles = folderTree.files || [];
            if (folderTree.folders) {
            }
            const existingFile = allFiles.find(file =>
            if (existingFile) {
            }
        }
    }
    try {
        if (creatingItem.type === 'folder') {
            // Extract language from extension
            const extension = name.split('.').pop()?.toLowerCase() || 'txt';
            const languageMap: Record<string, string> = {
            };
            const language = languageMap[extension] || 'javascript';
            const fileId = await createFile({
        }
        if (error instanceof Error && error.message.includes('already exists')) {
        }
    }
};
const handleCreateCancel = () => {
};
const handleDelete = async (type: 'folder' | 'file', id: string) => {
    if (!user?.id) return;
    const confirmMessage = type === 'folder'
    if (!confirm(confirmMessage)) return;
    try {
```

```
            if (type === 'folder') {
            }
        }
    };
    // eslint-disable-next-line @typescript-eslint/no-explicit-any
    const renderFolder = (folder: any, depth = 0) => {
        const isExpanded = expandedFolders.has(folder._id);
        const paddingClass = depth === 0 ? 'pl-2' : depth === 1 ? 'pl-6' : depth === 2 ? 'pl-10' : 'pl-14';
        // [JSX RENDERING CODE EXCLUDED]
    };
    // eslint-disable-next-line @typescript-eslint/no-explicit-any
    const renderFile = (file: any, depth = 0) => {
        const isSelected = selectedFileId === file._id;
        const paddingClass = depth === 0 ? 'pl-2' : depth === 1 ? 'pl-6' : depth === 2 ? 'pl-10' : 'pl-14';
        // [JSX RENDERING CODE EXCLUDED]
    };
    if (!folderTree) {
        // [JSX RENDERING CODE EXCLUDED]
    }
    // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/collaboration/MultiSessionFileEditor.tsx

Size: 16794 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■■ ***Business Logic*** *- Core application functionality*
■ *Contains: 9 imports, 1 exports, ~12 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*
■ *Integrates Monaco code editor*

■ Functions & Logic Analysis:

Function: handleKeyDown
• *Asynchronous function (uses async/await)*
• *Parameters: event*
• *Transforms/processes data arrays*
• *Handles user events/interactions*

```
    const handleKeyDown = async (event: KeyboardEvent) => {
      if (event.ctrlKey && event.key === 's') {
        if (activeTabId && user?.id) {
          const editorContent = unsavedChanges.get(activeTabId);
          if (editorContent !== undefined) {
            try {
              // Remove from unsaved changes
                const newMap = new Map(prev);
              // Update tab dirty state
                prevTabs.map(tab =>
            }
          }
        }
      }
    };
```

Function: closeTab
• *Parameters: fileId, event?*
• *Transforms/processes data arrays*
• *Performs validation checks*

```
  const closeTab = (fileId: Id<"sessionFiles">, event?: React.MouseEvent) => {
    if (event) {
    }
    // Check for unsaved changes
    const hasUnsavedChanges = unsavedChanges.has(fileId);
    if (hasUnsavedChanges) {
      const shouldClose = confirm("You have unsaved changes. Close anyway?");
      if (!shouldClose) return;
    }
    // Get remaining tabs before removing current one
    const remainingTabs = tabs.filter(tab => tab.fileId !== fileId);
    // Remove tab
    // Remove unsaved...
```

Function: handleEditorChange

- *Parameters: value*
- *Transforms/processes data arrays*
- *Handles user events/interactions*

```
  const handleEditorChange = (value: string | undefined) => {
    if (value === undefined || !activeTabId) return;
    // Store unsaved changes
    const newUnsavedChanges = new Map(unsavedChanges);
    // Mark tab as dirty
      prevTabs.map(tab =>
  };
```

Function: runCode

- *Asynchronous function (uses async/await)*
- *Makes HTTP/API requests*

```
  const runCode = async () => {
    if (!activeFile || !activeTabId) return;
    const code = unsavedChanges.get(activeTabId) ?? activeFile.code;
    if (!code.trim()) {
    }
    try {
      const runtime = LANGUAGE_CONFIG[activeFile.language]?.pistonRuntime;
      if (!runtime) {
      }
      const response = await fetch("https://emkc.org/api/v2/piston/execute", {
        },
      const data = await response.json();
      if (data.message) {
      }
      if (data.compile && data.compile.code...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState, useEffect, useCallback } from "react";
import { useQuery, useMutation } from "convex/react";
import { api } from "../../../convex/_generated/api";
import { useUser } from "@clerk/nextjs";
import { Editor } from "@monaco-editor/react";
import { defineMonacoThemes, LANGUAGE_CONFIG } from "@/app/(root)/_constants";
import { XIcon, FileIcon, PlayIcon, Loader2Icon } from "lucide-react";
import { Id } from "../../../convex/_generated/dataModel";
import useMounted from "@/hooks/useMounted";
interface EditorTab {
}
interface MultiSessionFileEditorProps {
}
export default function MultiSessionFileEditor({
  const { user } = useUser();
  const mounted = useMounted();
  const [tabs, setTabs] = useState<EditorTab[]>([]);
  const [activeTabId, setActiveTabId] = useState<Id<"sessionFiles"> | null>(null);
  const [unsavedChanges, setUnsavedChanges] = useState<Map<string, string>>(new Map());
  const [isRunning, setIsRunning] = useState(false);
  const [output, setOutput] = useState("");
```

```
const [error, setError] = useState<string | null>(null);
const [showOutput, setShowOutput] = useState(false);
// Helper function to deduplicate tabs
const deduplicateTabs = (tabs: EditorTab[]): EditorTab[] => {
  const seen = new Set<string>();
  return tabs.filter(tab => {
    if (seen.has(tab.fileId)) {
    }
};
// Apply deduplication whenever tabs change
useEffect(() => {
    const deduplicated = deduplicateTabs(prevTabs);
    if (deduplicated.length !== prevTabs.length) {
    }
// Mutations
const updateFile = useMutation(api.sessionFiles.updateFile);
// Queries
const activeFile = useQuery(
const openFile = useCallback((fileId: Id<"sessionFiles">) => {
  // Check if file is already open
    const existingTab = prevTabs.find(tab => tab.fileId === fileId);
    if (existingTab) {
    }
    // Create new tab with loading state
    const optimisticTab: EditorTab = {
    };
// Handle file selection from file tree
useEffect(() => {
  if (selectedFileId && user?.id) {
  }
// Keyboard shortcut for saving (Ctrl+S)
useEffect(() => {
  const handleKeyDown = async (event: KeyboardEvent) => {
    if (event.ctrlKey && event.key === 's') {
      if (activeTabId && user?.id) {
        const editorContent = unsavedChanges.get(activeTabId);
        if (editorContent !== undefined) {
          try {
            // Remove from unsaved changes
              const newMap = new Map(prev);
            // Update tab dirty state
              prevTabs.map(tab =>
          }
        }
      }
    }
  };
  // [JSX RENDERING CODE EXCLUDED]
  };
// Update tabs when activeFile data is loaded
useEffect(() => {
  if (activeFile && activeTabId) {
      const existingTabIndex = prevTabs.findIndex(tab => tab.fileId === activeTabId);
      if (existingTabIndex === -1) {
        // Tab doesn't exist, create new one
        const newTab: EditorTab = {
        };
        // Update existing tab
        const existingTab = prevTabs[existingTabIndex];
        if (existingTab.isLoading) {
          const updatedTabs = [...prevTabs];
          };
        }
      }
  }
const closeTab = (fileId: Id<"sessionFiles">, event?: React.MouseEvent) => {
  if (event) {
  }
  // Check for unsaved changes
  const hasUnsavedChanges = unsavedChanges.has(fileId);
  if (hasUnsavedChanges) {
    const shouldClose = confirm("You have unsaved changes. Close anyway?");
    if (!shouldClose) return;
  }
  // Get remaining tabs before removing current one
  const remainingTabs = tabs.filter(tab => tab.fileId !== fileId);
  // Remove tab
  // Remove unsaved changes
    const newMap = new Map(prev);
```

```
    // Update active tab
    if (activeTabId === fileId) {
      if (remainingTabs.length > 0) {
      }
    }
  };
  const handleEditorChange = (value: string | undefined) => {
    if (value === undefined || !activeTabId) return;
    // Store unsaved changes
    const newUnsavedChanges = new Map(unsavedChanges);
    // Mark tab as dirty
      prevTabs.map(tab =>
  };
  // Auto-save effect
  useEffect(() => {
    if (!activeTabId || !user?.id) return;
    const interval = setInterval(async () => {
      const editorContent = unsavedChanges.get(activeTabId);
      if (editorContent !== undefined) {
        try {
          // Remove from unsaved changes
          const newUnsavedChanges = new Map(unsavedChanges);
          // Update tab dirty state
            prevTabs.map(tab =>
        }
      }
    // [JSX RENDERING CODE EXCLUDED]
  // Run code function
  const runCode = async () => {
    if (!activeFile || !activeTabId) return;
    const code = unsavedChanges.get(activeTabId) ?? activeFile.code;
    if (!code.trim()) {
    }
    try {
      const runtime = LANGUAGE_CONFIG[activeFile.language]?.pistonRuntime;
      if (!runtime) {
      }
      const response = await fetch("https://emkc.org/api/v2/piston/execute", {
        },
      const data = await response.json();
      if (data.message) {
      }
      if (data.compile && data.compile.code !== 0) {
        const error = data.compile.stderr || data.compile.output;
      }
      if (data.run && data.run.code !== 0) {
        const error = data.run.stderr || data.run.output;
      }
      const output = data.run.output;
    }
  };
  if (!mounted) return null;
  // [JSX RENDERING CODE EXCLUDED]
}
```

■ File: src/components/collaboration/SavedSessions.tsx

Size: 12042 bytes | Last Modified: 2025-10-31 16:53:49

■ ***File Overview:***
■■ ***Business Logic*** *- Core application functionality*
■ *Contains: 9 imports, 1 exports, ~8 functions/constants*
■ *Uses Convex backend framework*
■ *Integrates Clerk authentication*

■ Functions & Logic Analysis:

Function: SavedSessions

• Performs database mutations (create/update/delete operations)
• Fetches data from the database
• Manages component state

```
export default function SavedSessions() {
  const { user } = useUser();
  const router = useRouter();
  const [editingSession, setEditingSession] = useState<string | null>(null);
  const [editData, setEditData] = useState({
  const [showDeleteConfirm, setShowDeleteConfirm] = useState<string | null>(null);
  // Queries
  const savedSessions = useQuery(
  const savedSessionInfo = useQuery(
  // Mutations
  const updateSavedSession = useMutation(api.collaboration.updateSavedSession);
  const delete...
```

■ Complete Code:

```
// DOCUMENTED BY SCRIPT - Phase 4
import { useState } from 'react';
import { useQuery, useMutation } from 'convex/react';
import { useUser } from '@clerk/nextjs';
import { useRouter } from 'next/navigation';
import { api } from '../../../convex/_generated/api';
import { Id } from '../../../convex/_generated/dataModel';
import { toast } from 'react-hot-toast';
import {
import { ConfirmModal } from '../ui/Modal';
export default function SavedSessions() {
  const { user } = useUser();
  const router = useRouter();
  const [editingSession, setEditingSession] = useState<string | null>(null);
  const [editData, setEditData] = useState({
  const [showDeleteConfirm, setShowDeleteConfirm] = useState<string | null>(null);
  // Queries
  const savedSessions = useQuery(
  const savedSessionInfo = useQuery(
  // Mutations
  const updateSavedSession = useMutation(api.collaboration.updateSavedSession);
  const deleteSavedSession = useMutation(api.collaboration.deleteSavedSession);
  const toggleSavedSessionPrivacy = useMutation(api.collaboration.toggleSavedSessionPrivacy);
  const startEditing = (session: { _id: string; name: string; description?: string; tags?: string[]; isPriva
  };
  const cancelEditing = () => {
  };
  const handleUpdate = async (sessionId: string) => {
    if (!user?.id) return;
    try {
    }
  };
  const handleDelete = async (sessionId: string) => {
    if (!user?.id) return;
    try {
    }
  };
  const formatDate = (timestamp: number) => {
  };
  const handleOpenSession = (sessionId: string) => {
  };
  const handleTogglePrivacy = async (sessionId: string) => {
    if (!user?.id) return;
    try {
      const result = await toggleSavedSessionPrivacy({
    }
  };
  // [JSX RENDERING CODE EXCLUDED]
}
```

Documentation Summary

All Phases Completed: Phase 1, 2, 3 & 4
Total Files Documented: 41
Files Processed:
• convex/schema.ts
• convex/_generated/dataModel.d.ts
• src/types/index.ts
• convex/auth.config.ts
• next.config.ts
• vercel.json
• tsconfig.json
• convex/codeExecution.ts
• convex/codeExecutions.ts
• convex/collaboration.ts
• convex/sessionActivity.ts
• convex/sessionFiles.ts
• convex/sessionFolders.ts
• convex/files.ts
• convex/folders.ts
• convex/snippets.ts
• convex/users.ts
• convex/lemonSqueezy.ts
• convex/http.ts
• convex/crons.ts
• convex/migration.ts
• src/middleware.ts
• src/app/api/session-leave/route.ts
• src/store/useCodeEditorStore.ts
• src/hooks/useSessionActivity.ts
• src/utils/sessionId.ts
• src/app/(root)/_components/EditorPanel.tsx
• src/app/(root)/_components/OutputPanel.tsx
• src/app/(root)/_components/ShareSnippetDialog.tsx
• src/app/(root)/_components/LanguageSelector.tsx
• src/app/(root)/_components/ThemeSelector.tsx
• src/components/MultiFileEditor.tsx
• src/components/MultiFileEditorSimple.tsx
• src/components/FileOperationsPanel.tsx
• src/components/FileTree.tsx
• src/components/collaboration/SessionManager.tsx
• src/components/collaboration/CollaborationIntegration.tsx
• src/components/collaboration/CollaborativeEditor.tsx
• src/components/collaboration/CollaborativeFileTree.tsx
• src/components/collaboration/MultiSessionFileEditor.tsx
• src/components/collaboration/SavedSessions.tsx