🧠 AI Context Vault – Architecture & Rulesets

🛠 Recommended Tech Stack & Project Setup

To create an optimal development environment for this Chrome extension with React:

✅ Core Technologies:

- **React** **+**
- **Webpack** for bundling and optimizing the extension
- **styled-components** or **Tailwind CSS** for styling (Tailwind recommended for rapid UI development)
- **React Hooks** for state management (useContext + useReducer pattern for global state)
- **Chrome Extension Manifest V** API integration
- **GitHub Gist API** for cross-device synchronization

✅ GitHub Gist Integration

Purpose: Provide seamless cross-device synchronization without requiring a custom backend service.

📐 Technical Implementation Details

Authentication Flow:

. **Initial OAuth Setup**:

- Register the extension with GitHub OAuth (callback: `chrome-extension://{extension-id}/options/auth-callback.html`)
- Request minimal scopes: `gist` only (for creating/reading private gists)
- Implement the OAuth . authorization code flow:

```
// Step 1: Redirect to GitHub OAuth
const authURL = `https://github.com/login/oauth/authorize?
         client_id=${CLIENT_ID}&scope=gist&state=$
         {secureRandomState}`;
chrome.tabs.create({ url: authURL });
```

```
// Step 2: Handle callback with code
// In auth-callback.html
const urlParams = new
        URLSearchParams(window.location.search);
const code = urlParams.get("code");
const state = urlParams.get("state");

// Verify state matches to prevent CSRF
if (state === storedState) {
  // Exchange code for token via background script proxy
  chrome.runtime.sendMessage({
    type: "EXCHANGE_GITHUB_CODE",
    code,
  });
}
```

. **Token Exchange and Storage**:

- Use a serverless function (Cloudflare Worker/Firebase Function) as proxy for exchanging the code:

```
// Background service worker
chrome.runtime.onMessage.addListener(async (message) => {
  if (message.type === "EXCHANGE_GITHUB_CODE") {
    // Call serverless endpoint to exchange code for token
    const response = await fetch(
      "https://your-serverless-fn.workers.dev/github-token",
      {
        method: "POST",
        body: JSON.stringify({ code: message.code }),
        headers: { "Content-Type": "application/json" },
      }
    );

    const { access_token } = await response.json();

    // Encrypt token before storage
    const encryptedToken = await encryptToken(access_token);

    // Store in chrome.storage.local
    await chrome.storage.local.set({
      github_token: encryptedToken,
      last_sync_timestamp: Date.now(),
    });
  }
});
```

. **Manual PAT Option**:

- Provide a fallback UI for users who prefer PAT:

```
const handlePATSubmit = async (pat) => {
  // Validate PAT format (40-character hex)
  if (!/^[0-9a-f]{40}$/.test(pat)) {
    setError("Invalid Personal Access Token format");
    return;
  }

  // Test token with a gist list API call
  try {
    const response = await fetch("https://api.github.com/
        gists", {
      headers: { Authorization: `token ${pat}` },
    });

    if (response.ok) {
      const encryptedToken = await encryptToken(pat);
      await chrome.storage.local.set({
        github_token: encryptedToken,
        token_type: "pat",
        last_sync_timestamp: Date.now(),
      });
      setSuccess("PAT verified and saved");
    } else {
      setError(
        "Token validation failed. Check permissions and try
        again."
      );
    }
  } catch (err) {
    setError(`Network error: ${err.message}`);
  }
};
```

. **Token Security**:

- Implement AES-GCM encryption using the browser’s Web Crypto API:

```
const encryptToken = async (token) => {
  // Generate device-specific encryption key from browser
        fingerprint
  const deviceKey = await generateDeviceKey();

  // Convert token to ArrayBuffer
  const encoder = new TextEncoder();
  const data = encoder.encode(token);

  // Generate random IV
  const iv = crypto.getRandomValues(new Uint8Array(12));

  // Encrypt
  const key = await crypto.subtle.importKey(
    "raw",
    deviceKey,
    { name: "AES-GCM" },
    false,
    ["encrypt"]
  );

  const ciphertext = await crypto.subtle.encrypt(
    { name: "AES-GCM", iv },
    key,
    data
  );

  // Combine IV and ciphertext for storage
  const result = new Uint8Array(iv.length +
        ciphertext.byteLength);
  result.set(iv);
  result.set(new Uint8Array(ciphertext), iv.length);

  // Convert to base64 for storage
  return btoa(String.fromCharCode(...new
        Uint8Array(result)));
};

const decryptToken = async (encryptedData) => {
  // Similar process in reverse
  // ...
};
```

Gist Structure and Data Format:

. **Main Gist Structure**:

```
{
  "description": "AI Context Vault Sync - Last updated:
        2023-06-15T20:45:12Z",
  "files": {
    "ai_context_vault_metadata.json": {
      "content": "{\"version\":\"1.2.0\",\"last_sync\":\"2023-06-15T20:45:12Z\"
        \"macbook-pro-work\",\"domains\":[\"chat.openai.com\",\"claude.ai\",
        \"gemini.google.com\"]}"
    },
    "chat.openai.com.json": {
      "content": "{\"chats\":[{\"chatId\":\"abc123\",
        \"summary\":\"Project X planning\",\"entries\":[...]},
        {\"chatId\":\"def456\",\"summary\":\"Bug analysis\",
        \"entries\":[...]}]}"
    },
    "claude.ai.json": {
      "content": "{\"chats\":[...]}"
    },
    "gemini.google.com.json": {
      "content": "{\"chats\":[...]}"
    }
  }
}
```

. **Individual Chat Entry Structure**:

```
{
  "chatId": "abc123xyz",
  "url": "https://chat.openai.com/c/abc123xyz",
  "title": "Project Planning Session",
  "summary": "AI assistant helping with project planning and
        task breakdown",
  "last_modified": "2023-06-15T20:42:11Z",
  "entries": [
    {
      "id": "entry_1686856931245",
      "text": "This project uses React 18 with TypeScript",
      "active": true,
      "created": "2023-06-15T20:22:11Z",
      "last_modified": "2023-06-15T20:22:11Z",
      "source": "user_selection",
      "metadata": {
        "selection_source": "assistant_message",
        "device_id": "macbook-pro-work"
      }
    },
    {
      "id": "entry_1686857021183",
      "text": "The deadline is June 30th, 2023",
      "active": true,
      "created": "2023-06-15T20:23:41Z",
      "last_modified": "2023-06-15T20:23:41Z",
      "source": "manual_entry",
        "metadata": {
          "device_id": "macbook-pro-work"
        }
      }
    ]
  }
```

. **Version Control and Change Tracking**:

◦ Include a `version_history` file to track sync operations:

```
{
  "version_history.json": {
    "content": "{\"history\":[{\"timestamp\":\"2023-06-15T20:45:12Z\",
        \"device_id\":\"macbook-pro-work\",\"operation\":\"sync\",
        \"changes\":{\"added\":2,\"modified\":1,\"deleted\":0}},
        {\"timestamp\":\"2023-06-14T10:12:33Z\",\"device_id\":\"iphone-
        personal\",\"operation\":\"sync\",\"changes\":{\"added\":1,
        \"modified\":0,\"deleted\":0}}]}"
  }
}
```

Sync Logic Implementation:

. **Initialization and First-Launch Flow**:

```
// Check for existing setup on extension first run
const initializeSync = async () => {
  const { github_token, gist_id } = await
        chrome.storage.local.get([
    "github_token",
    "gist_id",
  ]);

  if (github_token) {
    // Already authenticated
    if (gist_id) {
      // Fully configured, perform sync
      await performSync();
      return { status: "synced" };
    } else {
      // Need to create or connect to gist
      return { status: "need_gist_setup" };
    }
  } else {
    // First time setup, show onboarding
    return { status: "need_auth" };
  }
};
```

. **First-Time Gist Setup**:

```
const setupGist = async (options) => {
  const token = await getDecryptedToken();

  if (options.action === "create") {
    // Create new gist with initial data
    const localData = await getAllLocalContextData();
    const files = generateGistFiles(localData);

    const response = await fetch("https://api.github.com/gists",
      {
      method: "POST",
      headers: {
        Authorization: `token ${token}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        description: "AI Context Vault Sync - Auto-generated",
        public: false,
        files,
      }),
    });

    if (response.ok) {
      const gist = await response.json();
      await chrome.storage.local.set({
        gist_id: gist.id,
        last_sync_timestamp: Date.now(),
        sync_status: "success",
      });
      return { status: "created", gist_id: gist.id };
    } else {
      throw new Error("Failed to create gist");
    }
  } else if (options.action === "connect") {
    // Connect to existing gist
    const gistId = extractGistId(options.gistUrl);

    // Validate gist exists and is accessible
    const response = await fetch(`https://api.github.com/gists/$
      {gistId}`, {
      headers: { Authorization: `token ${token}` },
    });
    if (response.ok) {
      const gist = await response.json();

      // Check if it's our gist format
      const isValidFormat = validateGistFormat(gist);

      if (isValidFormat) {
        await chrome.storage.local.set({
          gist_id: gistId,
          last_sync_timestamp: Date.now(),
        });

        // Handle merging of remote and local data
        await handleDataMerge(gist);

        return { status: "connected", gist_id: gistId };
      } else {
        throw new Error("Invalid gist format");
      }
    } else {
      throw new Error("Failed to access gist");
    }
  }
};
```

. **Sync Queue Implementation**:

```
// In-memory queue with persistence
let syncQueue = [];
let syncTimeout = null;

const queueSync = async (changeType, data) => {
  // Add to queue
  syncQueue.push({
    type: changeType,
    data,
    timestamp: Date.now(),
  });

  // Persist queue to prevent loss on extension restart
  await chrome.storage.local.set({ sync_queue: syncQueue });

  // Clear any existing timeout
  if (syncTimeout) {
    clearTimeout(syncTimeout);
  }
```

```
    // Set new timeout (5s delay)
    syncTimeout = setTimeout(() => {
      processQueue();
    }, 5000);
  };

  const processQueue = async () => {
    if (syncQueue.length === 0) return;

    // Set sync status to indicate in progress
    await chrome.storage.local.set({ sync_status:
          "in_progress" });

    try {
      // Get latest from remote first
      await pullRemoteChanges();

      // Apply queued changes
      const changes = compactChanges(syncQueue);

      // Push changes to remote
      await pushChangesToRemote(changes);

      // Clear queue
      syncQueue = [];
      await chrome.storage.local.set({
        sync_queue: [],
        last_sync_timestamp: Date.now(),
        sync_status: "success",
      });
    } catch (error) {
      console.error("Sync failed:", error);
      await chrome.storage.local.set({
        sync_status: "failed",
        sync_error: error.message,
      });

      // Retry with exponential backoff
      scheduleRetry();
    }
  };
```

. **CTRL+I Integration**:

```
// After adding context
const handleAddContext = async (url, text) => {
  // First add locally
  const result = await contextStorage.addContext(url, text);

  // Show confirmation bubble
  showContextBubble(text, result.index);

  // Queue sync
  await queueSync("add", {
    url,
    entryId: result.entryId,
    chatId: result.chatId,
  });

  return result;
};
```

Data Merging Implementation:

. **Merge Strategy Functions**:

```
const handleDataMerge = async (remoteGist) => {
  // Extract local data
  const localData = await getAllLocalContextData();

  // Extract remote data
  const remoteData = parseGistData(remoteGist);

  // Compare and create merge plan
  const mergePlan = createMergePlan(localData, remoteData);

  if (mergePlan.hasConflicts) {
    // Store pending merge in local storage
    await chrome.storage.local.set({
      pending_merge: mergePlan,
      sync_status: "conflict",
    });

    // Show conflict UI in next appropriate moment
    return { status: "conflict", conflicts:
        mergePlan.conflicts.length };
  } else {
    // Auto-merge non-conflicting changes
    const mergedData = performAutoMerge(localData, remoteData,
        mergePlan);

    // Save merged data locally
    await saveAllContextData(mergedData);

    return {
      status: "auto_merged",
      stats: {
        added: mergePlan.add.length,
        updated: mergePlan.update.length,
        removed: mergePlan.remove.length,
      },
    };
  }
};

const createMergePlan = (localData, remoteData) => {
  const plan = {
    add: [], // Entries in remote not in local
    update: [], // Entries in both with remote being newer
    keep: [], // Entries in both with local being newer
    remove: [], // Entries in local not in remote (deletions)
    conflicts: [], // True conflicts needing resolution
    hasConflicts: false,
  };

  // Iterate through all domains and chats
  Object.keys(remoteData).forEach((domain) => {
    // Handle domain-level merging
    // ...

    // Handle chat-level merging
    remoteData[domain].chats.forEach((remoteChat) => {
      const localChat = findChatById(localData, domain,
        remoteChat.chatId);

      if (!localChat) {
        // New chat from remote, add all
        plan.add.push({
          type: "chat",
          domain,
          chat: remoteChat,
        });
      } else {
        // Compare entries
        remoteChat.entries.forEach((remoteEntry) => {
          const localEntry = findEntryById(localChat,
        remoteEntry.id);

          if (!localEntry) {
            // New entry from remote
            plan.add.push({
              type: "entry",
              domain,
              chatId: remoteChat.chatId,
      entry: remoteEntry,
    });
  } else {
    // Compare timestamps
    const remoteTime = new
Date(remoteEntry.last_modified).getTime();
    const localTime = new
Date(localEntry.last_modified).getTime();

    if (remoteTime > localTime) {
      // Remote is newer
      plan.update.push({
        type: "entry",
        domain,
        chatId: remoteChat.chatId,
        entry: remoteEntry,
        existing: localEntry,
      });
    } else if (remoteTime < localTime) {
      // Local is newer
      plan.keep.push({
        type: "entry",
        domain,
        chatId: remoteChat.chatId,
        entry: localEntry,
      });
    } else {
      // Same timestamp but different content
      if (
        remoteEntry.text !== localEntry.text ||
        remoteEntry.active !== localEntry.active
      ) {
        plan.conflicts.push({
          type: "entry",
          domain,
          chatId: remoteChat.chatId,
          remote: remoteEntry,
          local: localEntry,
        });
        plan.hasConflicts = true;
      }
    }
  }
});

// Check for local entries not in remote (potential
deletions)
localChat.entries.forEach((localEntry) => {
  const remoteEntry = findEntryById(remoteChat,
localEntry.id);
```

```
          if (!remoteEntry) {
            plan.remove.push({
              type: "entry",
              domain,
              chatId: localChat.chatId,
              entry: localEntry,
            });
          }
        });
      }
    });
  });

  return plan;
};
```

. **Conflict Resolution UI**:

```
// React component for conflict resolution
const ConflictResolver = ({ conflicts, onResolve }) => {
  const [resolutions, setResolutions] = useState({});

  const handleResolution = (conflictId, resolution) => {
    setResolutions((prev) => ({
      ...prev,
      [conflictId]: resolution, // 'local', 'remote', or 'both'
    }));
  };

  const applyResolutions = () => {
    onResolve(resolutions);
  };

  return (
    <div className="conflict-resolver">
      <h2>Sync Conflicts Detected</h2>
      <p>Please resolve the following conflicts:</p>

      {conflicts.map((conflict) => (
        <ConflictItem
          key={conflict.id}
          conflict={conflict}
          resolution={resolutions[conflict.id] || null}
          onResolve={(resolution) =>
            handleResolution(conflict.id, resolution)
          }
        />
        ))}

        <div className="actions">
          <button
            disabled={Object.keys(resolutions).length !==
          conflicts.length}
            onClick={applyResolutions}
          >
            Apply Resolutions
          </button>
          <button onClick={() => onResolve("keep_local_all")}>
            Keep All Local
          </button>
          <button onClick={() => onResolve("keep_remote_all")}>
            Keep All Remote
          </button>
        </div>
      </div>
    );
  };

  const ConflictItem = ({ conflict, resolution, onResolve }) => {
    return (
      <div className="conflict-item">
        <div className="conflict-header">
          <span className="domain">{conflict.domain}</span>
          <span className="chat-id">Chat: {conflict.chatId}</span>
        </div>

        <div className="conflict-content">
          <div className="local-version">
            <h4>Local Version</h4>
            <pre>{conflict.local.text}</pre>
            <div className="metadata">
              Last modified:
          {formatDate(conflict.local.last_modified)}
              {conflict.local.active ? "✅ Active" : "❌ Inactive"}
            </div>
          </div>

          <div className="remote-version">
            <h4>Remote Version</h4>
            <pre>{conflict.remote.text}</pre>
            <div className="metadata">
              Last modified:
          {formatDate(conflict.remote.last_modified)}
              {conflict.remote.active ? "✅ Active" : "❌
          Inactive"}
            </div>
          </div>
      </div>

      <div className="resolution-options">
        <button
          className={resolution === "local" ? "selected" : ""}
          onClick={() => onResolve("local")}
        >
          Keep Local
        </button>
        <button
          className={resolution === "remote" ? "selected" : ""}
          onClick={() => onResolve("remote")}
        >
          Keep Remote
        </button>
        <button
          className={resolution === "both" ? "selected" : ""}
          onClick={() => onResolve("both")}
        >
          Keep Both
        </button>
      </div>
    </div>
  );
};
```

Error Handling & Resilience:

. **Comprehensive Error Handling**:

```
const performSync = async () => {
  try {
    // Set sync status
    await updateSyncStatus("in_progress");

    // Get auth token
    const token = await
        getDecryptedToken().catch(handleAuthError);
    if (!token) return;

    // Get gist ID
    const { gist_id } = await
        chrome.storage.local.get("gist_id");
    if (!gist_id) {
      throw new SyncError("missing_gist_id", "No Gist ID
        configured");
    }
    // Fetch remote gist
    const gist = await fetchGist(token,
        gist_id).catch(handleNetworkError);
    if (!gist) return;

    // Process sync logic
    // ...

    // Update success status
    await updateSyncStatus("success");
  } catch (error) {
    // Categorize error
    let errorType = "unknown";

    if (error instanceof SyncError) {
      errorType = error.code;
    } else if (error.message.includes("rate limit")) {
      errorType = "rate_limit";
    } else if (error.message.includes("network")) {
      errorType = "network";
    } else if (error.message.includes("permission")) {
      errorType = "permissions";
    }

    // Handle based on type
    await handleSyncError(errorType, error);
  }
};

const handleSyncError = async (type, error) => {
  console.error(`Sync error (${type}):`, error);

  // Update status with error details
  await updateSyncStatus("error", {
    type,
    message: error.message,
    timestamp: Date.now(),
  });

  // Different handling based on error type
  switch (type) {
    case "rate_limit":
      // Schedule retry after rate limit window
      const retryAfter = error.headers?.["x-ratelimit-reset"]
        ? parseInt(error.headers["x-ratelimit-reset"]) * 1000
        : Date.now() + 60 * 60 * 1000; // Default 1 hour

      await scheduleRetry(retryAfter);
      break;
    case "network":
      // Exponential backoff for network issues
      await scheduleRetry(null, true);
      break;

    case "permissions":
    case "auth_expired":
      // Trigger re-auth flow
      await triggerReauth();
      break;

    case "missing_gist_id":
    case "invalid_gist":
      // Trigger gist setup
      await triggerGistSetup();
      break;

    default:
      // General retry with notification
      await scheduleRetry();
  }

  // Show user notification if appropriate
  if (["permissions", "auth_expired", "invalid_gist"].includes(type))
        {
    showSyncErrorNotification(type, error.message);
  }
};
```

. **Offline Queue Management**:

```
// Check for network availability
const isOnline = () => navigator.onLine;

// Monitor connectivity changes
window.addEventListener("online", handleOnline);
window.addEventListener("offline", handleOffline);

const handleOffline = () => {
  chrome.storage.local.set({ network_status: "offline" });
  // Pause any active sync operations
  if (syncTimeout) {
    clearTimeout(syncTimeout);
  }
};
```

```
const handleOnline = async () => {
  chrome.storage.local.set({ network_status: "online" });

  // Check if we have queued changes
  const { sync_queue, sync_status } = await
        chrome.storage.local.get([
    "sync_queue",
    "sync_status",
  ]);

  if (sync_queue?.length > 0 || sync_status ===
        "offline_pending") {
    // Process queue now that we're online
    processQueue();
  }
};
```

. **Sync Status Indicator Component**:

```
const SyncStatusIndicator = () => {
  const [status, setStatus] = useState("unknown");
  const [error, setError] = useState(null);
  const [lastSync, setLastSync] = useState(null);

  useEffect(() => {
    // Initial status load
    loadStatus();

    // Listen for status changes
    const listener = chrome.storage.onChanged.addListener((changes)
        => {
      if (
        changes.sync_status ||
        changes.last_sync_timestamp ||
        changes.sync_error
      ) {
        loadStatus();
      }
    });

    return () =>
        chrome.storage.onChanged.removeListener(listener);
  }, []);

  const loadStatus = async () => {
    const { sync_status, last_sync_timestamp, sync_error } =
      await chrome.storage.local.get([
        "sync_status",
        "last_sync_timestamp",
        "sync_error",
      ]);

    setStatus(sync_status || "unknown");
    setError(sync_error || null);
    setLastSync(last_sync_timestamp || null);
  };

  const getStatusIcon = () => {
    switch (status) {
      case "success":
        return "✅";
      case "in_progress":
        return "🔄";
      case "error":
        return "❌";
      case "offline_pending":
        return "📡";
      case "conflict":
        return "⚠️";
      default:
        return "❓";
    }
  };

  const handleManualSync = () => {
    chrome.runtime.sendMessage({ type: "MANUAL_SYNC" });
  };

  return (
    <div className={`sync-status ${status}`}>
      <span className="icon">{getStatusIcon()}</span>
      <span className="label">
        {status === "success" && "Synced"}
        {status === "in_progress" && "Syncing..."}
        {status === "error" && "Sync Error"}
        {status === "offline_pending" && "Offline - Changes Pending"}
        {status === "conflict" && "Sync Conflict"}
        {status === "unknown" && "Not Synced"}
      </span>
      {lastSync && (
        <span className="timestamp">
          Last: {formatRelativeTime(lastSync)}
        </span>
      )}
      <button
        onClick={handleManualSync}
        disabled={status === "in_progress"}
        title="Sync Now"
      >
        🔄
      </button>
      {error && <div className="error-details">{error}</div>}
    </div>
  );
};
```

Implementation Timeline and Dependencies:

. **Phase : Basic OAuth Flow and Storage**

- Implement GitHub OAuth flow
- Setup secure token storage
- Build PAT input alternative

. **Phase : Gist Operations and Data Format**

- Implement Gist create/read/update operations
- Define and validate data structures
- Create utility functions for data transformation

. **Phase : Sync Logic and Background Processes**

- Build sync queue system
- Implement background sync processes
- Add CTRL+I integration
- Build conflict detection

. **Phase : UI Components and User Experience**

- Develop first-time setup UI
- Create conflict resolution interface
- Implement sync status indicators
- Add settings page for sync preferences

. **Phase : Testing and Refinement**

- Comprehensive testing across devices
- Network condition simulations
- Edge case handling
- Performance optimization

✅ Project Structure (Extended):

```
ai-context-vault/
├── public/
│   ├── manifest.json
│   ├── icons/
│   └── background.js
├── src/
│   ├── components/
│   │   ├── ContextManager/
│   │   ├── ContextBubble/
│   │   └── shared/
│   ├── hooks/
│   │   └── useContextStorage.ts
│   ├── utils/
│   ├── services/
│   │   └── contextStorage.ts
│   ├── content.tsx
│   └── options.tsx
├── webpack.config.js
├── tsconfig.json
└── package.json
```

✅ Development Workflow:

- **webpack-dev-server** with hot reload support
- **Babel** for React/JSX compilation
- **ESLint** + **Prettier** for code quality
- **Jest** + **React Testing Library** for unit tests

🧩 Core Component Breakdown

✅ contextStorage.js

Purpose: Persistent storage of user-defined context entries per domain/chat session.

📐 Rules & Behaviors:

Storage Key Structure:

```
ctx_<hostname>_<pathHash>_<chatId>
```

- Prevents collision between similar tools (e.g., chat.openai.com vs claude.ai).
- Includes the current `chatId` (extracted from URL where available) to uniquely tie context to a specific conversation.

Enhanced Data Format:

```
{
  "chatId": "abc123xyz", // optional, used if determinable from URL
  "summary": "string",
  "entries": [
    { "text": "context line", "active": true, "created": timestamp }
  ]
}
```

Functions:

- `addContext(url, text)`
- `deleteContext(url, text)`
- `toggleContext(url, index)`
- `updateInitialSummary(url, summary)`
- `getContext(url)`
- `importContext(json)`
- `exportContext(url)` → JSON

✅ inject.js

Purpose: DOM listener and AI textbox enhancer.

📐 Rules & Logic:

Keyboard Shortcuts:

- `CTRL+I or CMD+I` : Save selected text to context
- `CTRL+J or CMD+J` : Open context manager overlay
- `ALT+ENTER` : Prepend context to message without sending
- `ALT+SHIFT+ENTER` : Inject context and send immediately

Target Textbox Detection:

- `textarea` or `div[contenteditable]` inside visible UI

Per Tool Detection Config:

```json
{
  "chat.openai.com": {
    "textboxSelector": "textarea",
    "sendButtonSelector": "button[aria-label=\"Send message\"]"
  },
  ...
}
```

On Send Intercept:

- Fetch saved context via `getContext()`
- Construct prepend:

```
[Summary]
- Context line 1
- Context line 2
...
---
[User prompt]
```

- Re-inject final message into textbox
- Optionally simulate Enter key or click send button

✅ ui-overlay.js

Purpose: Floating panel for context management (invoked with CTRL+SHIFT+I).

📐 Rules & Behaviors:

- Uses a fixed z-index to float above AI interface

Displays:

- Summary textbox
- List of context items with:
  - ✅/❌ toggle

    - ✏️ edit
    - 🗑️ delete

Buttons:

- ➕ Add New
- 🔁 Export To Clipboard (copy all JSON with message “Copied to clipboard”)
- 🔁 Import (show current raw JSON + allow user to paste replacement in our format)

Storage:

- All changes saved immediately via `contextStorage.js`

✅ background.js

Purpose: Central dispatcher for hotkeys + messaging bridge.

📐 Rules:

Listens for:

- `CTRL+I` → Send `save-selected-context` message to content script
- `CTRL+SHIFT+I` → Open overlay

Routing:

- Handles URL + chatId-specific routing to match context to current AI tool session

Save-Selected-Context Logic:

- When `CTRL+I` is pressed, the script checks for any user text currently selected across the page.
- This includes:
    - Text highlighted in the **prompt textbox** (textarea or contenteditable).
    - Text selected from **prior chat history**, including previous user prompts and AI responses (usually in structured divs/spans rendered by the AI platform).
- Uses `window.getSelection().toString()` to extract the exact visible text the user highlighted.

- Sends the highlighted string to the content script, which calls `addContext(url, selectedText)` using the current tab's URL and inferred chatId.
- If nothing is selected, a notification or fallback action can optionally alert the user.

Popup Confirmation Bubble:

- After successfully saving a context entry, display a non-intrusive popup bubble near the selection.
- The bubble should:
  - Appear with a subtle animation (fade-in or slide)
  - Show a checkmark icon with "Added to Context" message
  - Include an "Undo" button to immediately remove the entry
  - Auto-dismiss after   seconds
  - Use React Portal for rendering outside the normal DOM hierarchy
  - Position dynamically based on the selection coordinates
  - Be styled to match the overall theme and be visually distinct but not disruptive

```
// Example bubble component (React)
const ContextBubble = ({ text, onUndo, position }) => {
  return (
    <Portal>
      <BubbleContainer style={{ top: position.y, left: position.x }}>
        <CheckIcon />
        <Message>Added to Context</Message>
        <PreviewText>{text.substring(0, 30)}...</PreviewText>
        <UndoButton onClick={onUndo}>Undo</UndoButton>
      </BubbleContainer>
    </Portal>
  );
};
```

✅ options.html / js / css

Purpose: Full UI to bulk-manage contexts across all tools/domains.

📐 Future Enhancements:

- Search and edit across multiple domains

- Sync with cloud or GitHub Gist
- Backup & restore from local file

🧠 Context Prepend Logic

```
if (summary) prepend summary + "\n\n"
for each entry where active === true:
  prepend "• " + entry.text
add two newlines
append user message
```

🧠 Future Enhancements Ideas

- ✅ Sync context to a GitHub Gist via OAuth
  - Auto-sync after context changes
  - Conflict resolution UI
  - Diff visualization between local/remote versions
  - Selective sync for specific chat domains
  - Backup rotation (keep last N versions in separate files)
- ✅ Scheduled injection (e.g., rotate context every N minutes)
- ✅ AI-summarize context auto-dump feature
- ✅ OpenAI/Claude token visualizer bar
- ✅ Workspace Profiles per Project (optional, saved sets)
- ✅ Export/import functionality with shareable format
- ✅ Context templates for common scenarios