Game Development Document System - Development Guide

This guide provides step-by-step instructions for implementing the Game Development Document System Chrome extension. It covers environment setup, project structure, and implementation details for the first milestone.

Development Environment Setup

Prerequisites

- Node.js (v16 or later) and npm
- VSCode
- Chrome browser

Initial Setup

1. Create a new directory for the project:

```bash
mkdir game-doc-system
cd game-doc-system
```

2. Initialize npm project:

```bash
npm init -y
```

3. Install development dependencies:

```bash
npm install --save-dev typescript webpack webpack-cli ts-loader copy-webpack-plugin @types/chro
```

4. Create a `tsconfig.json` file:

json

```
{
  "compilerOptions": {
    "target": "es2020",
    "module": "es2020",
    "strict": true,
    "esModuleInterop": true,
    "outDir": "./dist",
    "moduleResolution": "node",
    "sourceMap": true,
    "lib": ["dom", "es2020"]
  },
  "include": ["src/**/*"]
}
```

5. Create a `webpack.config.js` file:

javascript

```javascript
const path = require('path');
const CopyPlugin = require('copy-webpack-plugin');

module.exports = {
  mode: 'development',
  entry: {
    background: './src/background/background.ts',
    content: './src/content/content.ts',
    popup: './src/popup/popup.ts'
  },
  devtool: 'source-map',
  module: {
    rules: [
      {
        test: /\.tsx?$/,
        use: 'ts-loader',
        exclude: /node_modules/
      }
    ]
  },
  resolve: {
    extensions: ['.tsx', '.ts', '.js']
  },
  output: {
    filename: '[name].js',
    path: path.resolve(__dirname, 'dist')
  },
  plugins: [
    new CopyPlugin({
      patterns: [
        { from: 'src/manifest.json', to: '.' },
        { from: 'src/popup/popup.html', to: '.' },
        { from: 'src/assets', to: '.' },
        { from: 'src/css', to: 'css' }
      ]
    })
  ]
};
```

6. Add scripts to `package.json`:

```json
"scripts": {
  "build": "webpack",
  "watch": "webpack --watch"
}
```

Project Structure

Create the following directory structure:

```
game-doc-system/
├── src/
│   ├── background/
│   │   └── background.ts
│   ├── content/
│   │   └── content.ts
│   ├── popup/
│   │   ├── popup.html
│   │   └── popup.ts
│   ├── shared/
│   │   ├── types.ts
│   │   └── storage.ts
│   ├── css/
│   │   └── sidebar.css
│   ├── assets/
│   │   ├── icon16.png
│   │   ├── icon48.png
│   │   └── icon128.png
│   └── manifest.json
├── dist/
├── tsconfig.json
├── webpack.config.js
└── package.json
```

Implementation Files

1. Manifest File (src/manifest.json)

json

```json
{
  "manifest_version": 3,
  "name": "Game Development Document System",
  "version": "0.1.0",
  "description": "Create game development documents with Claude AI",
  "icons": {
    "16": "assets/icon16.png",
    "48": "assets/icon48.png",
    "128": "assets/icon128.png"
  },
  "action": {
    "default_popup": "popup.html",
    "default_icon": {
      "16": "assets/icon16.png",
      "48": "assets/icon48.png",
      "128": "assets/icon128.png"
    }
  },
  "permissions": [
    "storage",
    "scripting"
  ],
  "host_permissions": [
    "https://claude.ai/*"
  ],
  "background": {
    "service_worker": "background.js",
    "type": "module"
  },
  "content_scripts": [
    {
      "matches": ["https://claude.ai/*"],
      "js": ["content.js"],
      "css": ["css/sidebar.css"]
    }
  ]
}
```

2. Types Definition (src/shared/types.ts)

typescript

```typescript
// Document types supported by the system
export enum DocumentType {
  GameVision = 'game_vision',
  CoreGameConcept = 'core_game_concept',
  TargetAudience = 'target_audience',
  CorePillarsValues = 'core_pillars_values',
  WhyPlayIt = 'why_play_it',
  WhatShouldTheyFeel = 'what_should_they_feel',
  UniqueSellingPoints = 'unique_selling_points',
  GameLoop = 'game_loop',
  PlayerJourney = 'player_journey',
  StoryOverview = 'story_overview',
  Presentation = 'presentation',
  KeyQuestions = 'key_questions',
  CoreDesignDetails = 'core_design_details',
  StrategicDirection = 'strategic_direction'
}

// Document status
export enum DocumentStatus {
  NotStarted = 'not_started',
  InProgress = 'in_progress',
  Completed = 'completed'
}

// Project structure
export interface GameProject {
  id: string;
  name: string;
  createdAt: number;
  updatedAt: number;
  documents: Record<string, GameDocument>;
}

// Document structure
export interface GameDocument {
  id: string;
  projectId: string;
  type: DocumentType;
  title: string;
  content: string;
  status: DocumentStatus;
  createdAt: number;
```

```
  updatedAt: number;
  versions: DocumentVersion[];
}

// Document version
export interface DocumentVersion {
  versionNumber: number;
  content: string;
  createdAt: number;
  notes: string;
}

// Message types for communication between components
export enum MessageType {
  GetProjects = 'get_projects',
  CreateProject = 'create_project',
  UpdateProject = 'update_project',
  DeleteProject = 'delete_project',
  GetProject = 'get_project',
  CreateDocument = 'create_document',
  UpdateDocument = 'update_document',
  GetDocument = 'get_document',
  DeleteDocument = 'delete_document',
  StartDocumentCreation = 'start_document_creation'
}

// Message structure
export interface Message {
  type: MessageType;
  payload: any;
}

// Response structure
export interface Response {
  success: boolean;
  data?: any;
  error?: string;
}
```

3. Storage Manager (src/shared/storage.ts)

typescript

```typescript
import { GameProject, GameDocument, DocumentType, DocumentStatus } from './types';

export class StorageManager {
  // Initialize storage
  public async initialize(): Promise<void> {
    const existingData = await this.getProjects();

    if (!existingData) {
      await chrome.storage.local.set({ projects: {} });
      console.log('Storage initialized with empty projects');
    }
  }

  // Get all projects
  public async getProjects(): Promise<Record<string, GameProject> | null> {
    const data = await chrome.storage.local.get('projects');
    return data.projects || null;
  }

  // Get specific project
  public async getProject(projectId: string): Promise<GameProject | null> {
    const projects = await this.getProjects();
    return projects && projects[projectId] ? projects[projectId] : null;
  }

  // Create new project
  public async createProject(name: string): Promise<GameProject> {
    const projects = await this.getProjects() || {};

    const projectId = 'project_' + Date.now();
    const timestamp = Date.now();

    const newProject: GameProject = {
      id: projectId,
      name,
      createdAt: timestamp,
      updatedAt: timestamp,
      documents: this.createInitialDocuments(projectId)
    };

    projects[projectId] = newProject;
    await chrome.storage.local.set({ projects });
```

```
    return newProject;
  }

  // Create the initial document set for a new project
  private createInitialDocuments(projectId: string): Record<string, GameDocument> {
    const documents: Record<string, GameDocument> = {};
    const timestamp = Date.now();

    // Create a document entry for each document type
    Object.values(DocumentType).forEach((type, index) => {
      const documentId = `${projectId}_${type}`;

      documents[documentId] = {
        id: documentId,
        projectId,
        type: type as DocumentType,
        title: this.getDefaultTitle(type as DocumentType),
        content: '',
        status: DocumentStatus.NotStarted,
        createdAt: timestamp,
        updatedAt: timestamp,
        versions: []
      };
    });

    return documents;
  }

  // Get default title for each document type
  private getDefaultTitle(type: DocumentType): string {
    const titles: Record<DocumentType, string> = {
      [DocumentType.GameVision]: 'Game Vision',
      [DocumentType.CoreGameConcept]: 'Core Game Concept',
      [DocumentType.TargetAudience]: 'Target Audience',
      [DocumentType.CorePillarsValues]: 'Core Pillars and Values',
      [DocumentType.WhyPlayIt]: 'Why Would They Play It?',
      [DocumentType.WhatShouldTheyFeel]: 'What Should They Feel?',
      [DocumentType.UniqueSellingPoints]: 'Unique Selling Points',
      [DocumentType.GameLoop]: 'Game Loop',
      [DocumentType.PlayerJourney]: 'Player Journey',
      [DocumentType.StoryOverview]: 'Story Overview',
      [DocumentType.Presentation]: 'Presentation',
      [DocumentType.KeyQuestions]: 'Key Questions',
      [DocumentType.CoreDesignDetails]: 'Core Design Details',
```

```
    [DocumentType.StrategicDirection]: 'Strategic Direction'
  };

  return titles[type] || 'Untitled Document';
}

// Update document
public async updateDocument(document: GameDocument): Promise<GameDocument> {
  const projects = await this.getProjects();
  if (!projects) throw new Error('No projects found');

  const project = projects[document.projectId];
  if (!project) throw new Error('Project not found');

  // Create new version if content changed
  const existingDoc = project.documents[document.id];
  if (existingDoc && existingDoc.content !== document.content && existingDoc.content !== '')
    const newVersion: DocumentVersion = {
      versionNumber: existingDoc.versions.length + 1,
      content: existingDoc.content,
      createdAt: Date.now(),
      notes: 'Automatic version created'
    };

    document.versions = [...existingDoc.versions, newVersion];
  }

  document.updatedAt = Date.now();
  project.documents[document.id] = document;
  project.updatedAt = Date.now();

  await chrome.storage.local.set({ projects });

  return document;
}

// Delete project
public async deleteProject(projectId: string): Promise<void> {
  const projects = await this.getProjects();
  if (!projects) return;

  delete projects[projectId];
  await chrome.storage.local.set({ projects });
}
```

```
}

// Create a singleton instance
export const storageManager = new StorageManager();
```

4. Background Script (src/background/background.ts)

typescript

```typescript
import { StorageManager, storageManager } from '../shared/storage';
import { MessageType, Response } from '../shared/types';

// Initialize when extension is installed or updated
chrome.runtime.onInstalled.addListener(async () => {
  console.log('Game Development Document System installed/updated');
  await storageManager.initialize();
});

// Listen for messages from content scripts or popup
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
  handleMessage(message)
    .then(sendResponse)
    .catch(error => {
      console.error('Error handling message:', error);
      sendResponse({ success: false, error: error.message });
    });

  // Return true to indicate we'll respond asynchronously
  return true;
});

// Handle messages based on their type
async function handleMessage(message: any): Promise<Response> {
  console.log('Background received message:', message);

  const { type, payload } = message;

  try {
    switch (type) {
      case MessageType.GetProjects:
        const projects = await storageManager.getProjects();
        return { success: true, data: projects };

      case MessageType.CreateProject:
        const project = await storageManager.createProject(payload.name);
        return { success: true, data: project };

      case MessageType.GetProject:
        const foundProject = await storageManager.getProject(payload.projectId);
        return { success: true, data: foundProject };

      case MessageType.UpdateDocument:
```

```
        const updatedDoc = await storageManager.updateDocument(payload.document);
        return { success: true, data: updatedDoc };

      case MessageType.DeleteProject:
        await storageManager.deleteProject(payload.projectId);
        return { success: true };

      default:
        return { success: false, error: `Unknown message type: ${type}` };
    }
  } catch (error) {
    return {
      success: false,
      error: error instanceof Error ? error.message : 'Unknown error'
    };
  }
}
```

5. Content Script (src/content/content.ts)

typescript

```
import { MessageType, GameProject, DocumentType, DocumentStatus } from '../shared/types';

// Wait until page fully loaded
window.addEventListener('load', () => {
  // Check if we're on Claude.ai
  if (window.location.href.includes('claude.ai')) {
    console.log('Game Development Document System loaded on Claude.ai');

    // Give the UI time to render completely
    setTimeout(injectSidebar, 2000);
  }
});

// Inject the sidebar into Claude's UI
function injectSidebar() {
  console.log('Attempting to inject sidebar');

  // Find container to inject into
  const claudeContainer = document.querySelector('.claude-container');
  if (!claudeContainer) {
    console.log('Claude container not found, retrying in 2 seconds');
    setTimeout(injectSidebar, 2000);
    return;
  }

  // Create sidebar element
  const sidebar = document.createElement('div');
  sidebar.id = 'gdds-sidebar';
  sidebar.className = 'gdds-sidebar';

  // Initial HTML for sidebar
  sidebar.innerHTML = `
    <div class="gdds-sidebar-header">
      <h3>Game Documents</h3>
      <button id="gdds-new-project" class="gdds-button primary">New Project</button>
      <button id="gdds-toggle-sidebar" class="gdds-button icon-only">
        <span class="gdds-icon">◀</span>
      </button>
    </div>
    <div class="gdds-projects-container">
      <div class="gdds-loading">Loading projects...</div>
    </div>
  `;
```

```
  // Add to DOM
  claudeContainer.appendChild(sidebar);

  // Setup event listeners
  setupEventListeners();

  // Load projects
  loadProjects();

  console.log('Sidebar injected successfully');
}

// Setup event listeners for sidebar
function setupEventListeners() {
  // New project button
  const newProjectButton = document.getElementById('gdds-new-project');
  if (newProjectButton) {
    newProjectButton.addEventListener('click', showNewProjectDialog);
  }

  // Toggle sidebar button
  const toggleButton = document.getElementById('gdds-toggle-sidebar');
  if (toggleButton) {
    toggleButton.addEventListener('click', toggleSidebar);
  }
}

// Toggle sidebar visibility
function toggleSidebar() {
  const sidebar = document.getElementById('gdds-sidebar');
  if (sidebar) {
    sidebar.classList.toggle('collapsed');

    // Update toggle button icon
    const toggleButton = document.getElementById('gdds-toggle-sidebar');
    if (toggleButton) {
      const icon = toggleButton.querySelector('.gdds-icon');
      if (icon) {
        icon.textContent = sidebar.classList.contains('collapsed') ? '▶' : '◀';
      }
    }
  }
}
```

```
// Show dialog to create new project
function showNewProjectDialog() {
  // Create dialog element
  const dialog = document.createElement('div');
  dialog.className = 'gdds-dialog';
  dialog.innerHTML = `
    <div class="gdds-dialog-content">
      <h3>Create New Project</h3>
      <form id="gdds-new-project-form">
        <div class="gdds-form-group">
          <label for="project-name">Project Name:</label>
          <input type="text" id="project-name" required>
        </div>
        <div class="gdds-form-actions">
          <button type="button" class="gdds-button secondary" id="gdds-cancel-project">Cancel</
          <button type="submit" class="gdds-button primary">Create Project</button>
        </div>
      </form>
    </div>
  `;

  // Add to DOM
  document.body.appendChild(dialog);

  // Focus the input field
  const nameInput = document.getElementById('project-name') as HTMLInputElement;
  if (nameInput) {
    nameInput.focus();
  }

  // Setup form submission
  const form = document.getElementById('gdds-new-project-form');
  if (form) {
    form.addEventListener('submit', (e) => {
      e.preventDefault();

      if (nameInput && nameInput.value.trim()) {
        createProject(nameInput.value.trim());
        closeDialog();
      }
    });
  }
```

```
  // Setup cancel button
  const cancelButton = document.getElementById('gdds-cancel-project');
  if (cancelButton) {
    cancelButton.addEventListener('click', closeDialog);
  }

  // Close dialog function
  function closeDialog() {
    document.body.removeChild(dialog);
  }
}

// Create new project
function createProject(name: string) {
  chrome.runtime.sendMessage(
    { type: MessageType.CreateProject, payload: { name } },
    (response) => {
      if (response.success) {
        console.log('Project created:', response.data);
        loadProjects();
      } else {
        console.error('Failed to create project:', response.error);
        showNotification('Error creating project', 'error');
      }
    }
  );
}

// Load projects from storage
function loadProjects() {
  const projectsContainer = document.querySelector('.gdds-projects-container');
  if (projectsContainer) {
    projectsContainer.innerHTML = '<div class="gdds-loading">Loading projects...</div>';

    chrome.runtime.sendMessage(
      { type: MessageType.GetProjects, payload: {} },
      (response) => {
        if (response.success) {
          renderProjects(response.data || {});
        } else {
          console.error('Failed to load projects:', response.error);
          projectsContainer.innerHTML = '<div class="gdds-error">Error loading projects</div>';
        }
      }
```

```
    );
  }
}

// Render projects list
function renderProjects(projects: Record<string, GameProject>) {
  const projectsContainer = document.querySelector('.gdds-projects-container');
  if (!projectsContainer) return;

  if (Object.keys(projects).length === 0) {
    projectsContainer.innerHTML = `
      <div class="gdds-empty-state">
        <p>No projects yet</p>
        <p>Click "New Project" to get started</p>
      </div>
    `;
    return;
  }

  // Create projects list
  const projectsList = document.createElement('div');
  projectsList.className = 'gdds-projects-list';

  // Sort projects by creation date (newest first)
  const sortedProjects = Object.values(projects).sort((a, b) =>
    b.createdAt - a.createdAt
  );

  // Add each project
  sortedProjects.forEach(project => {
    const projectElement = document.createElement('div');
    projectElement.className = 'gdds-project-item';
    projectElement.dataset.projectId = project.id;

    projectElement.innerHTML = `
      <div class="gdds-project-header">
        <h4>${project.name}</h4>
        <div class="gdds-project-actions">
          <button class="gdds-button icon-only project-expand">
            <span class="gdds-icon">▼</span>
          </button>
        </div>
      </div>
      <div class="gdds-project-documents" style="display: none;"></div>
```

```
    `;

    projectsList.appendChild(projectElement);

    // Add event listener for expand/collapse
    const expandButton = projectElement.querySelector('.project-expand');
    if (expandButton) {
      expandButton.addEventListener('click', () => {
        toggleProjectDocuments(project.id);
      });
    }
  });

  // Replace loading indicator with projects list
  projectsContainer.innerHTML = '';
  projectsContainer.appendChild(projectsList);
}

// Toggle project documents visibility
function toggleProjectDocuments(projectId: string) {
  const projectElement = document.querySelector(`.gdds-project-item[data-project-id="${projectI
  if (!projectElement) return;

  const documentsContainer = projectElement.querySelector('.gdds-project-documents');
  if (!documentsContainer) return;

  const isVisible = documentsContainer.style.display !== 'none';
  documentsContainer.style.display = isVisible ? 'none' : 'block';

  // Update expand/collapse icon
  const expandButton = projectElement.querySelector('.project-expand');
  if (expandButton) {
    const icon = expandButton.querySelector('.gdds-icon');
    if (icon) {
      icon.textContent = isVisible ? '▼' : '▲';
    }
  }

  // If expanding, load and render documents
  if (!isVisible) {
    loadProjectDocuments(projectId);
  }
}
```

```
// Load document list for a project
function loadProjectDocuments(projectId: string) {
  chrome.runtime.sendMessage(
    { type: MessageType.GetProject, payload: { projectId } },
    (response) => {
      if (response.success && response.data) {
        renderProjectDocuments(response.data);
      } else {
        console.error('Failed to load project documents:', response.error);
      }
    }
  );
}

// Render documents for a project
function renderProjectDocuments(project: GameProject) {
  const projectElement = document.querySelector(`.gdds-project-item[data-project-id="${project.
  if (!projectElement) return;

  const documentsContainer = projectElement.querySelector('.gdds-project-documents');
  if (!documentsContainer) return;

  // Clear previous content
  documentsContainer.innerHTML = '';

  // Define the document type order
  const documentTypeOrder = [
    DocumentType.GameVision,
    DocumentType.CoreGameConcept,
    DocumentType.TargetAudience,
    DocumentType.CorePillarsValues,
    DocumentType.WhyPlayIt,
    DocumentType.WhatShouldTheyFeel,
    DocumentType.UniqueSellingPoints,
    DocumentType.GameLoop,
    DocumentType.PlayerJourney,
    DocumentType.StoryOverview,
    DocumentType.Presentation,
    DocumentType.KeyQuestions,
    DocumentType.CoreDesignDetails,
    DocumentType.StrategicDirection
  ];

  // Sort documents by the defined order
```

```
const sortedDocuments = Object.values(project.documents).sort((a, b) => {
  return documentTypeOrder.indexOf(a.type as DocumentType) -
         documentTypeOrder.indexOf(b.type as DocumentType);
});

// Create document list
sortedDocuments.forEach(document => {
  const documentElement = document.createElement('div');
  documentElement.className = `gdds-document-item status-${document.status}`;
  documentElement.dataset.documentId = document.id;

  // Status text based on document status
  const statusText = {
    [DocumentStatus.NotStarted]: 'Not Started',
    [DocumentStatus.InProgress]: 'In Progress',
    [DocumentStatus.Completed]: 'Completed'
  }[document.status] || 'Unknown';

  documentElement.innerHTML = `
    <div class="gdds-document-info">
      <span class="gdds-document-title">${document.title}</span>
      <span class="gdds-document-status">${statusText}</span>
    </div>
    <div class="gdds-document-actions">
      ${document.status === DocumentStatus.NotStarted
        ? '<button class="gdds-button small create-doc">Create</button>'
        : '<button class="gdds-button small view-doc">View</button>'}
      ${document.status === DocumentStatus.Completed
        ? '<button class="gdds-button small icon-only export-doc"><span class="gdds-icon">⬇️
        : ''}
    </div>
  `;

  documentsContainer.appendChild(documentElement);

  // Add event listeners for document actions
  const createButton = documentElement.querySelector('.create-doc');
  if (createButton) {
    createButton.addEventListener('click', () => {
      startDocumentCreation(project.id, document.id);
    });
  }

  const viewButton = documentElement.querySelector('.view-doc');
```

```
    if (viewButton) {
      viewButton.addEventListener('click', () => {
        viewDocument(project.id, document.id);
      });
    }

    const exportButton = documentElement.querySelector('.export-doc');
    if (exportButton) {
      exportButton.addEventListener('click', () => {
        exportDocument(project.id, document.id);
      });
    }
  });
}

// Start document creation process
function startDocumentCreation(projectId: string, documentId: string) {
  console.log(`Starting document creation: Project ${projectId}, Document ${documentId}`);

  // Will be implemented in next phase - for now just show notification
  showNotification('Document creation will be implemented in the next phase', 'info');
}

// View document
function viewDocument(projectId: string, documentId: string) {
  console.log(`Viewing document: Project ${projectId}, Document ${documentId}`);

  // Will be implemented in next phase - for now just show notification
  showNotification('Document viewing will be implemented in the next phase', 'info');
}

// Export document
function exportDocument(projectId: string, documentId: string) {
  console.log(`Exporting document: Project ${projectId}, Document ${documentId}`);

  // Will be implemented in next phase - for now just show notification
  showNotification('Document export will be implemented in the next phase', 'info');
}

// Show notification
function showNotification(message: string, type: 'success' | 'error' | 'info' = 'info') {
  const notification = document.createElement('div');
  notification.className = `gdds-notification ${type}`;
  notification.innerHTML = `
```

```
      <div class="gdds-notification-content">
        <span class="gdds-notification-message">${message}</span>
        <button class="gdds-notification-close">×</button>
      </div>
    `;

    // Add to DOM
    document.body.appendChild(notification);

    // Add close button listener
    const closeButton = notification.querySelector('.gdds-notification-close');
    if (closeButton) {
      closeButton.addEventListener('click', () => {
        document.body.removeChild(notification);
      });
    }

    // Auto-remove after 5 seconds
    setTimeout(() => {
      if (document.body.contains(notification)) {
        document.body.removeChild(notification);
      }
    }, 5000);
  }
```

6. CSS Styling (src/css/sidebar.css)

css

```css
.gdds-sidebar {
  position: fixed;
  top: 0;
  right: 0;
  width: 300px;
  height: 100vh;
  background-color: #2a2a2a;
  color: #f0f0f0;
  z-index: 1000;
  box-shadow: -2px 0 10px rgba(0, 0, 0, 0.2);
  font-family: -apple-system, BlinkMacSystemFont, sans-serif;
  transition: transform 0.3s ease;
  display: flex;
  flex-direction: column;
}

.gdds-sidebar.collapsed {
  transform: translateX(290px);
}

.gdds-sidebar-header {
  padding: 15px;
  display: flex;
  align-items: center;
  justify-content: space-between;
  border-bottom: 1px solid #3e3e3e;
}

.gdds-sidebar-header h3 {
  margin: 0;
  font-size: 16px;
  font-weight: 600;
}

.gdds-projects-container {
  flex: 1;
  overflow-y: auto;
  padding: 10px 0;
}

.gdds-projects-list {
  display: flex;
  flex-direction: column;
```

```
  gap: 10px;
}

.gdds-project-item {
  background-color: #363636;
  border-radius: 6px;
  margin: 0 10px;
  overflow: hidden;
}

.gdds-project-header {
  padding: 12px 15px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  cursor: pointer;
}

.gdds-project-header h4 {
  margin: 0;
  font-size: 14px;
  font-weight: 500;
}

.gdds-project-documents {
  padding: 10px;
  background-color: #2f2f2f;
  display: flex;
  flex-direction: column;
  gap: 8px;
}

.gdds-document-item {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 8px 12px;
  border-radius: 4px;
  font-size: 13px;
  background-color: #404040;
}

.gdds-document-item.status-not_started {
  opacity: 0.7;
```

```
}

.gdds-document-item.status-in_progress {
  border-left: 3px solid #ffc107;
}

.gdds-document-item.status-completed {
  border-left: 3px solid #4caf50;
}

.gdds-document-info {
  display: flex;
  flex-direction: column;
  gap: 2px;
}

.gdds-document-title {
  font-weight: 500;
}

.gdds-document-status {
  font-size: 11px;
  opacity: 0.7;
}

.gdds-document-actions {
  display: flex;
  gap: 5px;
}

.gdds-button {
  background-color: #525252;
  color: #ffffff;
  border: none;
  border-radius: 4px;
  padding: 8px 12px;
  font-size: 13px;
  cursor: pointer;
  transition: background-color 0.2s;
}

.gdds-button:hover {
  background-color: #676767;
}
```

```css
.gdds-button.primary {
  background-color: #8c52ff;
}

.gdds-button.primary:hover {
  background-color: #9d6aff;
}

.gdds-button.secondary {
  background-color: transparent;
  border: 1px solid #525252;
}

.gdds-button.secondary:hover {
  background-color: rgba(255, 255, 255, 0.1);
}

.gdds-button.small {
  padding: 4px 8px;
  font-size: 12px;
}

.gdds-button.icon-only {
  padding: 4px 6px;
  display: flex;
  align-items: center;
  justify-content: center;
}

.gdds-icon {
  font-size: 14px;
  line-height: 1;
}

.gdds-empty-state {
  padding: 20px;
  text-align: center;
  opacity: 0.7;
  font-size: 14px;
}

.gdds-loading {
  padding: 20px;
```

```
  text-align: center;
  font-size: 14px;
}

.gdds-error {
  padding: 20px;
  text-align: center;
  color: #ff6b6b;
  font-size: 14px;
}

.gdds-dialog {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 2000;
}

.gdds-dialog-content {
  background-color: #2a2a2a;
  border-radius: 8px;
  padding: 20px;
  width: 400px;
  max-width: 90%;
}

.gdds-dialog-content h3 {
  margin-top: 0;
  font-size: 18px;
}

.gdds-form-group {
  margin-bottom: 15px;
}

.gdds-form-group label {
  display: block;
  margin-bottom: 5px;
```

```
  font-size: 14px;
}

.gdds-form-group input {
  width: 100%;
  padding: 8px 10px;
  border-radius: 4px;
  border: 1px solid #4a4a4a;
  background-color: #333333;
  color: #f0f0f0;
  font-size: 14px;
}

.gdds-form-actions {
  display: flex;
  justify-content: flex-end;
  gap: 10px;
  margin-top: 20px;
}

.gdds-notification {
  position: fixed;
  bottom: 20px;
  right: 320px;
  background-color: #333333;
  border-radius: 6px;
  padding: 12px 15px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.2);
  z-index: 1500;
  transition: transform 0.3s, opacity 0.3s;
  animation: gdds-slide-in 0.3s;
  max-width: 300px;
}

.gdds-notification.success {
  border-left: 4px solid #4caf50;
}

.gdds-notification.error {
  border-left: 4px solid #ff5252;
}

.gdds-notification.info {
  border-left: 4px solid #2196f3;
```

```css
}

.gdds-notification-content {
  display: flex;
  align-items: center;
  gap: 10px;
}

.gdds-notification-message {
  flex: 1;
  font-size: 14px;
}

.gdds-notification-close {
  background: none;
  border: none;
  color: #999;
  cursor: pointer;
  font-size: 16px;
  padding: 0;
}

@keyframes gdds-slide-in {
  from {
    transform: translateX(50px);
    opacity: 0;
  }
  to {
    transform: translateX(0);
    opacity: 1;
  }
}
```

7. Popup HTML (src/popup/popup.html)

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Game Development Document System</title>
  <style>
    body {
      font-family: -apple-system, BlinkMacSystemFont, sans-serif;
      width: 300px;
      margin: 0;
      padding: 15px;
      background-color: #2a2a2a;
      color: #f0f0f0;
    }

    h1 {
      font-size: 16px;
      margin-top: 0;
    }

    p {
      font-size: 14px;
      line-height: 1.5;
    }

    .button {
      display: inline-block;
      background-color: #8c52ff;
      color: white;
      border: none;
      padding: 8px 12px;
      border-radius: 4px;
      font-size: 14px;
      cursor: pointer;
      text-decoration: none;
      margin-top: 10px;
    }

    .button:hover {
      background-color: #9d6aff;
    }
```

```
    .status {
      margin-top: 15px;
      padding: 10px;
      background-color: #363636;
      border-radius: 4px;
    }

    .status-label {
      font-weight: bold;
      margin-bottom: 5px;
    }

    .project-count {
      margin-top: 5px;
    }
  </style>
</head>
<body>
  <h1>Game Development Document System</h1>

  <p>Create, manage, and export game design documents with Claude AI.</p>

  <div class="status">
    <div class="status-label">Status:</div>
    <div id="connection-status">Checking connection...</div>
    <div class="project-count" id="project-count"></div>
  </div>

  <button id="open-claude" class="button">Open Claude.ai</button>

  <script src="popup.js"></script>
</body>
</html>
```

8. Popup Script (src/popup/popup.ts)

typescript

```
import { MessageType } from '../shared/types';

// When popup loads
document.addEventListener('DOMContentLoaded', () => {
  // Check connection status
  checkConnectionStatus();

  // Load project count
  loadProjectCount();

  // Setup open Claude button
  const openClaudeButton = document.getElementById('open-claude');
  if (openClaudeButton) {
    openClaudeButton.addEventListener('click', () => {
      chrome.tabs.create({ url: 'https://claude.ai' });
    });
  }
});

// Check if we're connected to Claude.ai
function checkConnectionStatus() {
  const statusElement = document.getElementById('connection-status');
  if (!statusElement) return;

  chrome.tabs.query({ url: 'https://claude.ai/*' }, (tabs) => {
    if (tabs.length > 0) {
      statusElement.textContent = 'Connected to Claude.ai';
      statusElement.style.color = '#4caf50'; // Green
    } else {
      statusElement.textContent = 'Not connected to Claude.ai';
      statusElement.style.color = '#ff9800'; // Orange
    }
  });
}

// Get project count
function loadProjectCount() {
  const countElement = document.getElementById('project-count');
  if (!countElement) return;

  chrome.runtime.sendMessage(
    { type: MessageType.GetProjects, payload: {} },
    (response) => {
```

```
      if (response.success) {
        const projects = response.data || {};
        const count = Object.keys(projects).length;

        countElement.textContent = `Projects: ${count}`;
      } else {
        countElement.textContent = 'Error loading projects';
      }
    }
  );
}
```

Loading the Extension in Chrome

1. Run the build command to compile the TypeScript code:

```bash
npm run build
```

2. Open Chrome and navigate to `chrome://extensions/`
3. Enable "Developer mode" by toggling the switch in the top-right corner
4. Click "Load unpacked" and select the `dist` folder from your project directory
5. The extension should now appear in your extensions list
6. Navigate to Claude.ai to see your sidebar in action

Next Steps

After implementing this basic structure, the next steps would be:

1. Implement the document creation process with Claude
2. Add the ability to view and edit documents
3. Implement document export functionality
4. Add Google Drive integration

Testing and Debugging

You can use the following for debugging:

- Chrome's developer tools to inspect the extension's behavior
- Console logs in background and content scripts

- Chrome's extension management page to inspect storage

Remember to reload the extension (in chrome://extensions/) after making changes to your code and rebuilding.