Technical Design Document: AI PII Sanitizer Chrome Extension

Document Overview

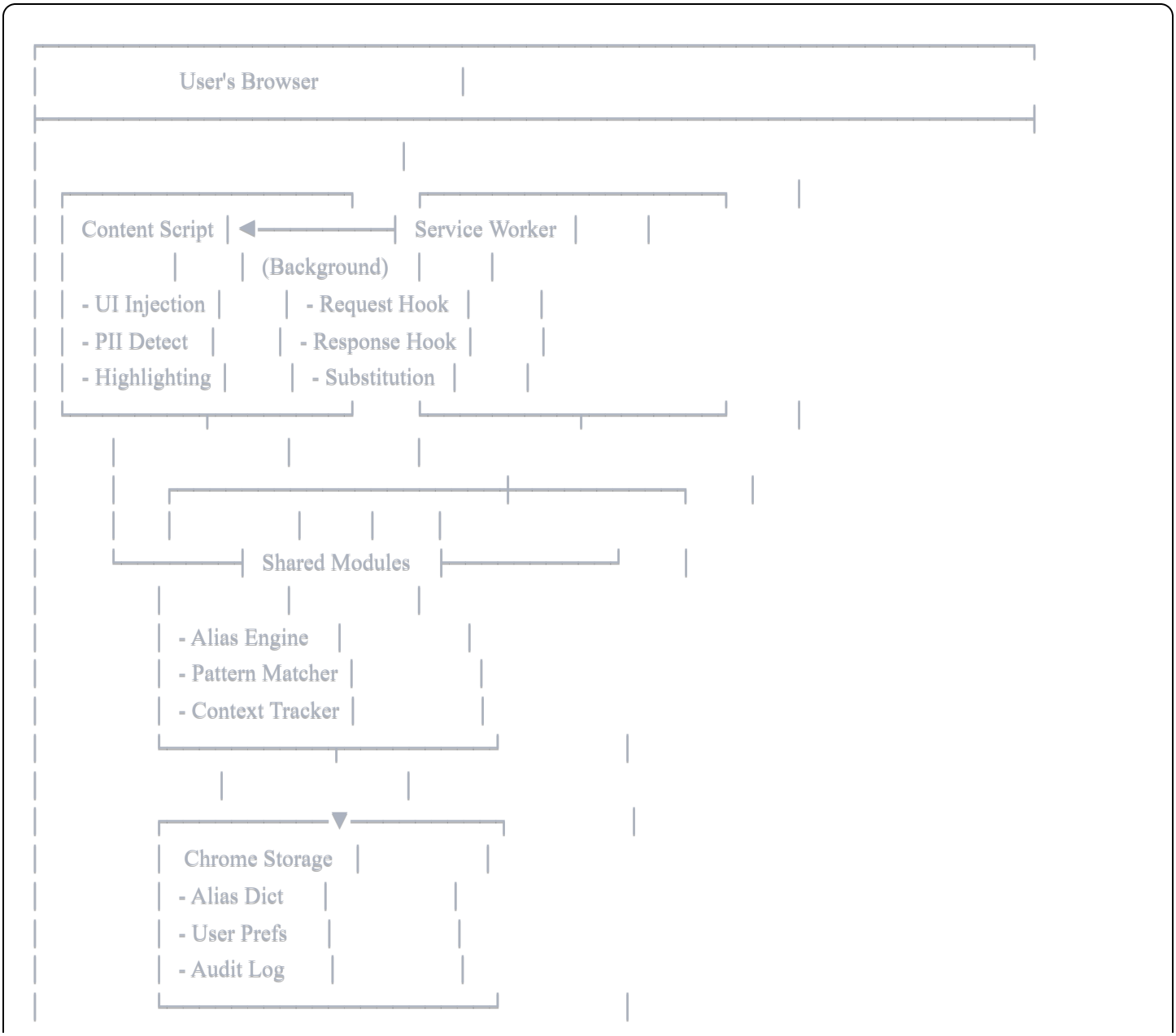
Purpose: Detailed technical specification for implementing a Chrome extension that performs bidirectional PII aliasing for AI chat services.

Audience: Engineers implementing the extension

Related Documents: Product Design Document (PDD)

System Architecture

High-Level Components



Component Responsibilities

Content Script (`content.js`):

- Injected into AI chat pages
- Monitors text input fields for PII
- Provides visual feedback (highlighting)
- Renders reversed responses
- Communicates with Service Worker

Service Worker (`background.js`):

- Intercepts fetch/XHR requests to AI domains
- Applies alias substitution to outgoing requests
- Intercepts responses, reverses aliases
- Manages Chrome Storage operations
- Coordinates between components

Alias Engine (`aliasEngine.js`):

- Core substitution logic
- Bidirectional mapping
- Context-aware replacement
- Handles edge cases (possessives, pronouns)

Pattern Matcher (`patternMatcher.js`):

- Efficient string matching (Aho-Corasick algorithm)
- Regex fallbacks for complex patterns
- Case-insensitive matching with case preservation

Popup UI (`popup.html/js`):

- Alias dictionary CRUD operations
- Statistics dashboard
- Enable/disable toggle
- Settings management

Data Structures

Alias Dictionary

```javascript
```

```
// Primary storage format
interface AliasEntry {
  id: string;              // UUID
  realValue: string;       // "Joe Smith"
  aliasValue: string;      // "John Doe"
  type: 'name' | 'email' | 'phone' | 'address'; // Future: expand types
  category?: string;       // "family", "work", "medical"
  metadata: {
    createdAt: number;
    usageCount: number;
    lastUsed: number;
    confidence: number;    // 0-1, for auto-detected aliases
  };
  enabled: boolean;
}

// In-memory lookup structures (built from AliasEntry[])
class AliasMap {
  private realToAlias: Map<string, string>;
  private aliasToReal: Map<string, string>;
  private patterns: Map<string, RegExp>; // For partial matching

  constructor(entries: AliasEntry[]) {
    // Build efficient lookup maps
    // Create regex patterns for context-aware matching
  }

  substitute(text: string, direction: 'real-to-alias' | 'alias-to-real'): string {
    // Core substitution logic
  }
}
```

Configuration Schema

javascript

```
interface UserConfig {
  version: number;
  settings: {
    enabled: boolean;
    autoHighlight: boolean;
    showNotifications: boolean;
    protectedDomains: string[];  // AI service domains
    excludedDomains: string[];   // Never intercept
    strictMode: boolean;         // Fail-safe: reject if PII detected but no alias
  };
  aliases: AliasEntry[];
  stats: {
    totalSubstitutions: number;
    successRate: number;
    lastSyncTimestamp: number;
  };
}
```

Context Tracking

javascript

```
interface ConversationContext {
  conversationId: string;  // Derived from URL or generated
  aliases: Set<string>;    // Aliases used in this conversation
  entities: Map<string, {  // AI-generated entities to track
    alias: string;
    firstMention: number;
    confidence: 'known' | 'inferred' | 'unknown';
  }>;
  history: {
    timestamp: number;
    direction: 'request' | 'response';
    substitutions: Array<{from: string, to: string}>;
  }[];
}
```

Core Algorithms

1. Substitution Engine

Goal: Replace all instances of real PII with aliases (or vice versa) while preserving:

- Case (Joe → John, joe → john, JOE → JOHN)
- Possessives (Joe's → John's)
- Context (surrounding punctuation, whitespace)

Implementation:

```javascript
```

```
class SubstitutionEngine {
  private aliasMap: AliasMap;

  /**
   * Main substitution function
   * Uses Aho-Corasick for efficient multi-pattern matching
   * Falls back to regex for complex cases
   */
  substitute(text: string, direction: 'encode' | 'decode'): SubstitutionResult {
    const map = direction === 'encode' ?
      this.aliasMap.realToAlias :
      this.aliasMap.aliasToReal;

    // Phase 1: Exact matches (fastest)
    let result = this.exactReplace(text, map);

    // Phase 2: Possessive forms
    result = this.handlePossessives(result, map);

    // Phase 3: Context-aware (only if needed)
    if (this.hasUnresolvedReferences(result)) {
      result = this.resolvePronouns(result, map);
    }

    return {
      text: result,
      substitutions: this.getSubstitutionLog(),
      confidence: this.calculateConfidence()
    };
  }

  /**
   * Exact string replacement preserving case
   */
  private exactReplace(text: string, map: Map<string, string>): string {
    // Sort by length (longest first) to handle overlapping matches
    const sortedKeys = Array.from(map.keys())
      .sort((a, b) => b.length - a.length);

    let result = text;
    for (const key of sortedKeys) {
      const replacement = map.get(key);
      // Use word boundaries to avoid partial matches
```

```
    const regex = new RegExp(`\\b${this.escapeRegex(key)}\\b`, 'gi');

    result = result.replace(regex, (match) => {
      // Preserve case of original match
      return this.preserveCase(match, replacement);
    });
  }
  return result;
}

/**
 * Preserve case pattern from original to replacement
 * "JOE SMITH" -> "JOHN DOE"
 * "Joe Smith" -> "John Doe"
 * "joe smith" -> "john doe"
 */
private preserveCase(original: string, replacement: string): string {
  if (original === original.toUpperCase()) {
    return replacement.toUpperCase();
  }
  if (original === original.toLowerCase()) {
    return replacement.toLowerCase();
  }
  // Title case: capitalize first letter of each word
  return replacement.split(' ').map((word, i) =>
    original.split(' ')[i]?.[0] === original.split(' ')[i]?.[0].toUpperCase()
      ? word.charAt(0).toUpperCase() + word.slice(1).toLowerCase()
      : word.toLowerCase()
  ).join(' ');
}

/**
 * Handle possessive forms: "Joe's car" -> "John's car"
 */
private handlePossessives(text: string, map: Map<string, string>): string {
  for (const [key, value] of map) {
    const possessivePattern = new RegExp(`\\b${this.escapeRegex(key)}'s\\b`, 'gi');
    text = text.replace(possessivePattern, `${value}'s`);
  }
  return text;
}
}
```

2. Request Interception (Service Worker)

Challenge: Chrome Manifest V3 restricts blocking webRequest. Solution: Use `declarativeNetRequest` with dynamic rules OR use fetch interception.

Approach: Fetch API interception using Service Worker

javascript

```
// background.js (Service Worker)

// Listen for messages from content script
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {
  if (message.type === 'INTERCEPT_REQUEST') {
    handleRequestIntercept(message.payload)
      .then(sendResponse);
    return true; // Keep channel open for async response
  }
});

/**
 * Content script proxies fetch through background script
 * Background script applies substitution before forwarding
 */
async function handleRequestIntercept(payload) {
  const { url, method, body, headers } = payload;

  // Parse body (usually JSON)
  let requestData;
  try {
    requestData = JSON.parse(body);
  } catch (e) {
    return { error: 'Cannot parse request body' };
  }

  // Extract text content (differs by AI service)
  const textContent = extractTextFromRequest(requestData, url);

  // Apply substitution (real → alias)
  const aliasEngine = await AliasEngine.getInstance();
  const substituted = aliasEngine.substitute(textContent, 'encode');

  // Reconstruct request body
  const modifiedRequestData = injectTextIntoRequest(
    requestData,
    substituted.text,
    url
  );

  // Forward modified request
  const response = await fetch(url, {
    method,
```

```
    headers,
    body: JSON.stringify(modifiedRequestData)
  });

  // Intercept response
  const responseData = await response.json();
  const responseText = extractTextFromResponse(responseData, url);

  // Apply reverse substitution (alias → real)
  const decoded = aliasEngine.substitute(responseText, 'decode');

  // Return modified response to content script
  return {
    success: true,
    modifiedResponse: injectTextIntoResponse(
      responseData,
      decoded.text,
      url
    )
  };
}

/**
 * Service-specific text extraction
 * Each AI service has different JSON structure
 */
function extractTextFromRequest(data, url) {
  if (url.includes('api.openai.com')) {
    // ChatGPT: messages array
    return data.messages?.map(m => m.content).join('\n') || '';
  } else if (url.includes('claude.ai')) {
    // Claude: prompt field
    return data.prompt || '';
  } else if (url.includes('gemini.google.com')) {
    // Gemini: contents array
    return data.contents?.map(c => c.parts?.map(p => p.text).join('')).join('\n') || '';
  }
  return '';
}
```

3. Content Script Integration

Challenge: Intercept fetch/XHR without modifying page code

Solution: Inject fetch wrapper early via content script

```javascript
```

```
// content.js - Injected into page

(function() {
  // Store original fetch
  const originalFetch = window.fetch;

  // Intercept fetch calls to AI APIs
  window.fetch = async function(...args) {
    const [url, options] = args;

    // Check if this is an AI API request
    if (shouldIntercept(url)) {
      // Send to background script for substitution
      const result = await chrome.runtime.sendMessage({
        type: 'INTERCEPT_REQUEST',
        payload: {
          url: url.toString(),
          method: options?.method || 'GET',
          body: options?.body,
          headers: options?.headers
        }
      });

      if (result.success) {
        // Return mocked response with substituted data
        return new Response(
          JSON.stringify(result.modifiedResponse),
          {
            status: 200,
            headers: { 'Content-Type': 'application/json' }
          }
        );
      }
    }

    // Pass through if not intercepting
    return originalFetch.apply(this, args);
  };

  // Detect AI service
  function shouldIntercept(url) {
    const aiDomains = [
      'api.openai.com',
```

```
    'claude.ai/api',
    'gemini.google.com/api'
  ];
  return aiDomains.some(domain => url.includes(domain));
 }
})();
```

4. Input Field Highlighting

Goal: Show user which text will be sanitized as they type

javascript

```
// content.js

class PIIHighlighter {
  constructor() {
    this.aliasEngine = null;
    this.observers = new Map();
  }

  async init() {
    this.aliasEngine = await AliasEngine.getInstance();
    this.observeInputFields();
  }

  observeInputFields() {
    // Monitor for new text input fields
    const observer = new MutationObserver((mutations) => {
      mutations.forEach(mutation => {
        mutation.addedNodes.forEach(node => {
          if (node.nodeType === 1) { // Element node
            this.attachToInputs(node);
          }
        });
      });
    });

    observer.observe(document.body, {
      childList: true,
      subtree: true
    });

    // Attach to existing inputs
    this.attachToInputs(document.body);
  }

  attachToInputs(root) {
    const inputs = root.querySelectorAll('textarea, input[type="text"], [contenteditable="true"]');
    inputs.forEach(input => {
      input.addEventListener('input', this.handleInput.bind(this));
    });
  }

  handleInput(event) {
    const element = event.target;
```

```
    const text = element.value || element.textContent;

    // Find PII matches
    const matches = this.aliasEngine.findPII(text);

    // Highlight matches (visual feedback)
    this.highlightMatches(element, matches);

    // Show tooltip with alias
    if (matches.length > 0) {
      this.showTooltip(element, matches);
    }
  }

  highlightMatches(element, matches) {
    // For contenteditable, wrap matches in <mark>
    // For textarea/input, show adjacent overlay with highlights
    // (textarea/input don't support internal HTML, need overlay div)

    if (element.contentEditable === 'true') {
      // Direct highlighting in contenteditable
      let html = element.innerHTML;
      matches.forEach(match => {
        const regex = new RegExp(`\\b${match.text}\\b`, 'gi');
        html = html.replace(regex, `<mark class="pii-highlight">${match.text}</mark>`);
      });
      element.innerHTML = html;
    } else {
      // Overlay approach for textarea
      this.createOverlay(element, matches);
    }
  }
}
```

Technology Stack

Core Technologies

- **Manifest V3**: Latest Chrome extension API
- **TypeScript**: Type safety, better DX
- **Webpack**: Bundle modules

- **Chrome Storage API**: Persistent data storage
- **Web Workers**: Offload heavy processing (optional)

Libraries

Pattern Matching:

- Custom Aho-Corasick implementation (lightweight)
- Alternative: `fuzzyset.js` for fuzzy matching (future)

UI Framework (Popup):

- Vanilla JS + Web Components (keep it light)
- Alternative: Preact (React-like, 3KB)

Testing:

- Jest: Unit tests
- Puppeteer: E2E tests
- Chrome Extension Testing Library

Build Pipeline

```bash
# Project structure
extension/
├── src/
│   ├── background/
│   │   └── serviceWorker.ts
│   ├── content/
│   │   └── content.ts
│   ├── popup/
│   │   ├── popup.html
│   │   └── popup.ts
│   ├── lib/
│   │   ├── aliasEngine.ts
│   │   ├── patternMatcher.ts
│   │   └── storage.ts
│   └── manifest.json
├── tests/
├── webpack.config.js
└── package.json
```

Build command:

```bash
npm run build  # Webpack bundles → dist/
```

Storage Strategy

Chrome Storage API

```javascript
```

```typescript
// lib/storage.ts

export class StorageManager {
  private static KEYS = {
    ALIASES: 'aliases',
    CONFIG: 'config',
    STATS: 'stats',
    AUDIT_LOG: 'audit_log'
  };

  /**
   * Save alias dictionary
   * Encrypted before storage for security
   */
  async saveAliases(aliases: AliasEntry[]): Promise<void> {
    const encrypted = await this.encrypt(JSON.stringify(aliases));
    await chrome.storage.local.set({
      [StorageManager.KEYS.ALIASES]: encrypted
    });
  }

  /**
   * Load and decrypt aliases
   */
  async loadAliases(): Promise<AliasEntry[]> {
    const data = await chrome.storage.local.get(StorageManager.KEYS.ALIASES);
    if (!data[StorageManager.KEYS.ALIASES]) {
      return [];
    }
    const decrypted = await this.decrypt(data[StorageManager.KEYS.ALIASES]);
    return JSON.parse(decrypted);
  }

  /**
   * Simple encryption using Web Crypto API
   * Key derived from extension ID (unique per install)
   */
  private async encrypt(data: string): Promise<string> {
    const encoder = new TextEncoder();
    const dataBuffer = encoder.encode(data);

    const key = await this.getEncryptionKey();
    const iv = crypto.getRandomValues(new Uint8Array(12));
```

```
    const encrypted = await crypto.subtle.encrypt(
      { name: 'AES-GCM', iv },
      key,
      dataBuffer
    );

    // Combine IV + encrypted data
    const combined = new Uint8Array(iv.length + encrypted.byteLength);
    combined.set(iv);
    combined.set(new Uint8Array(encrypted), iv.length);

    return this.arrayBufferToBase64(combined);
  }

  private async decrypt(encryptedData: string): Promise<string> {
    const combined = this.base64ToArrayBuffer(encryptedData);
    const iv = combined.slice(0, 12);
    const data = combined.slice(12);

    const key = await this.getEncryptionKey();
    const decrypted = await crypto.subtle.decrypt(
      { name: 'AES-GCM', iv },
      key,
      data
    );

    return new TextDecoder().decode(decrypted);
  }
}
```

Performance Considerations

Optimization Targets

Operation	Target	Notes
Substitution (1KB text)	<10ms	Real-time typing feedback
Request interception	<50ms	Imperceptible to user
Response processing	<100ms	Acceptable latency
Storage write	<200ms	Background operation

Optimization Strategies

1. **Lazy Loading**: Load alias dictionary once, keep in memory
2. **Memoization**: Cache substitution results for repeated text
3. **Web Workers**: Offload heavy regex to worker thread
4. **Incremental Updates**: Only re-process changed portions of text
5. **Throttling/Debouncing**: Limit highlight updates during typing

```javascript
// Example: Throttled input handler
const handleInput = throttle((event) => {
  const text = event.target.value;
  highlightPII(text);
}, 300); // Update every 300ms max
```

Security Considerations

Threat Model

Threats:

1. **Alias dictionary theft**: Attacker gains access to storage, learns real identities
2. **Man-in-the-middle**: Attacker intercepts traffic between extension and AI service
3. **Extension tampering**: Malicious code injected into extension
4. **Side-channel leaks**: Timing attacks, usage patterns reveal PII

Mitigations:

1. **Encrypt Storage**:
   - Use Web Crypto API (AES-GCM)
   - Derive key from extension ID + device ID
   - Never store key in plaintext
2. **Secure Communication**:
   - All AI services use HTTPS (browser enforces)
   - Extension never transmits aliases to external servers
   - No telemetry by default

3. **Code Integrity**:
    - Sign extension builds
    - Use Subresource Integrity (SRI) for any CDN scripts
    - Regular security audits
4. **Minimize Permissions**:
    - Only request necessary host permissions
    - Use `activeTab` instead of `<all_urls>` where possible
    - Explain each permission in privacy policy

Privacy Policy

Key Points:

- No data collection (zero telemetry in MVP)
- Aliases stored locally only
- No third-party scripts
- Open source for transparency
- Opt-in cloud sync (future)

Testing Strategy

Unit Tests

```javascript
```

```
// tests/aliasEngine.test.ts

describe('AliasEngine', () => {
  let engine;

  beforeEach(() => {
    const aliases = [
      { realValue: 'Joe Smith', aliasValue: 'John Doe', type: 'name' },
      { realValue: 'Sarah Chen', aliasValue: 'Emma Wilson', type: 'name' }
    ];
    engine = new AliasEngine(aliases);
  });

  test('substitutes single name', () => {
    const result = engine.substitute('Hello Joe Smith', 'encode');
    expect(result.text).toBe('Hello John Doe');
  });

  test('preserves case', () => {
    const result = engine.substitute('JOE SMITH is here', 'encode');
    expect(result.text).toBe('JOHN DOE is here');
  });

  test('handles possessives', () => {
    const result = engine.substitute("Joe Smith's car", 'encode');
    expect(result.text).toBe("John Doe's car");
  });

  test('bidirectional substitution', () => {
    const encoded = engine.substitute('Meet Joe Smith', 'encode');
    const decoded = engine.substitute(encoded.text, 'decode');
    expect(decoded.text).toBe('Meet Joe Smith');
  });

  test('handles overlapping names', () => {
    // Edge case: "John Smith" where "John" is part of alias "John Doe"
    // Should not partially substitute
  });
});
```

Integration Tests

javascript

```
// tests/e2e/chatgpt.test.ts

describe('ChatGPT Integration', () => {
  let browser, page, extensionId;

  beforeAll(async () => {
    // Launch Chrome with extension loaded
    browser = await puppeteer.launch({
      headless: false,
      args: [`--load-extension=./dist`]
    });

    // Get extension ID and navigate to popup
    extensionId = await getExtensionId(browser);
  });

  test('intercepts and substitutes ChatGPT request', async () => {
    // Set up alias
    await setAlias('Joe Smith', 'John Doe');

    // Navigate to ChatGPT
    page = await browser.newPage();
    await page.goto('https://chat.openai.com');

    // Type message containing real name
    await page.type('[data-testid="chat-input"]', 'Tell me about Joe Smith');
    await page.click('[data-testid="send-button"]');

    // Intercept network request
    const request = await page.waitForRequest(req =>
      req.url().includes('api.openai.com/v1/chat')
    );

    const body = JSON.parse(request.postData());
    const message = body.messages[0].content;

    // Verify substitution occurred
    expect(message).toContain('John Doe');
    expect(message).not.toContain('Joe Smith');
  });

  test('reverses alias in response', async () => {
    // ... similar test for response direction
```

```
  });
});
```

Manual Test Scenarios

1. **Basic flow**: Create alias → Send prompt → Verify substitution → Check response
2. **Edge cases**:
   - Empty alias dictionary
   - Special characters in names
   - Very long names (100+ chars)
   - Unicode names (Chinese, Arabic, emoji)
3. **Performance**: 10KB prompt with 50 PII instances
4. **Error handling**: Network failure during substitution
5. **Cross-browser**: Test on Chrome, Edge, Brave

Implementation Roadmap

Phase 1: Core MVP (Weeks 1-4)

Week 1: Foundation

- [ ] Set up project structure (TypeScript + Webpack)
- [ ] Implement manifest.json (V3)
- [ ] Basic storage manager
- [ ] Simple popup UI (add/remove aliases)

Week 2: Substitution Engine

- [ ] Alias engine core logic
- [ ] Pattern matching (exact matches only)
- [ ] Unit tests for substitution
- [ ] Case preservation

Week 3: Request Interception

- [ ] Service worker setup
- [ ] ChatGPT API interception
- [ ] Request body parsing
- [ ] Response interception

Week 4: Integration

- [ ] Content script injection
- [ ] End-to-end flow working
- [ ] Basic error handling
- [ ] Manual testing

Phase 2: Polish (Weeks 5-6)

Week 5: UX Improvements

- [ ] Input field highlighting
- [ ] Visual feedback (notifications)
- [ ] Improved popup UI
- [ ] Onboarding flow

Week 6: Additional AI Services

- [ ] Claude integration
- [ ] Gemini integration
- [ ] Settings page (enable/disable per service)
- [ ] Error recovery UI

Phase 3: Beta Release (Week 7-8)

Week 7: Quality & Security

- [ ] Comprehensive testing
- [ ] Security audit
- [ ] Performance optimization
- [ ] Documentation

Week 8: Launch

- [ ] Chrome Web Store submission
- [ ] GitHub repository public
- [ ] Landing page
- [ ] Initial user outreach

Open Questions & Future Work

Open Questions for MVP

1. **Context Window**: How many previous messages to consider for pronoun resolution?
2. **Error UX**: Show errors inline vs notification vs popup?
3. **Performance**: Is real-time highlighting too expensive? Add delay?
4. **Permissions**: Request `<all_urls>` or specific domains?

Post-MVP Features

Phase 2: Advanced Aliasing

- [ ] Email addresses, phone numbers, addresses
- [ ] Relationship preservation ("my wife Sarah" → maintain relationship)
- [ ] Auto-suggest aliases based on context
- [ ] Import/export alias dictionary

Phase 3: Enterprise Features

- [ ] Team shared dictionaries
- [ ] SSO integration
- [ ] Audit logs and compliance reports
- [ ] Admin controls (enforce sanitization)

Phase 4: Advanced Capabilities

- [ ] OCR for uploaded images
- [ ] Voice input sanitization
- [ ] Multi-language support
- [ ] AI-assisted alias generation

Appendix: API Reference

Message Protocol (Content Script ↔ Service Worker)

```typescript
```

```typescript
// Request types
type MessageType =
  | 'INTERCEPT_REQUEST'
  | 'GET_ALIASES'
  | 'ADD_ALIAS'
  | 'REMOVE_ALIAS'
  | 'UPDATE_CONFIG';

interface Message {
  type: MessageType;
  payload: any;
}

// Example: Intercept request
{
  type: 'INTERCEPT_REQUEST',
  payload: {
    url: string;
    method: string;
    body: string;
    headers: Record<string, string>;
  }
}

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

Storage Schema

typescript

```
// chrome.storage.local structure
{
  "aliases": string,  // Encrypted JSON of AliasEntry[]
  "config": {
    "version": 1,
    "enabled": true,
    "settings": { ... }
  },
  "stats": {
    "totalSubstitutions": number,
    "lastUsed": timestamp
  }
}
```

Document Version: 1.0

Last Updated: October 2025

Status: Ready for Implementation