HTML 6.x - Der Paradigmenwechsel zum Weaving-Internet

Itheereum 'Sear' Interface und GAIA OS im Kontext von WordPress Full-Site-Editing (FSE)

Von Gemini Deep Research und Stanislaus Kroppach, Berlin (EU) 18.11.2025

1. Einleitung: Der Paradigmenwechsel zum "Weaving"-Internet

Der vorliegende Forschungsbericht stellt eine umfassende, detailorientierte Analyse der technischen Herausforderungen, strategischen Implikationen und implementierungstechnischen Lösungen dar, die für die erfolgreiche Integration der "Itheereum Sear App" in die bestehende WordPress-Infrastruktur erforderlich sind. Basierend auf den bereitgestellten Dokumentationen – insbesondere der Strategie zur modularen Architektur [1], dem CSS-Kompendium für "Sear-Emoji" [1] und dem Konzeptpapier zum "Weaving" Media-Player [1] – adressiert dieser Bericht nicht nur die unmittelbaren Hindernisse bei der Template-Erkennung, sondern kontextualisiert diese innerhalb der breiteren Vision eines "HTML 6.x" oder "Weaving-Internets".

Die Analyse erfolgt vor dem Hintergrund einer fundamentalen Transformation des Content-Management-Systems WordPress, das sich derzeit in einer kritischen Übergangsphase von der klassischen PHP-basierten Theme-Architektur hin zur JSON-gesteuerten Full-Site-Editing (FSE) Logik befindet. Diese Transition erzeugt spezifische Friktionen – wie das Verschwinden von PHP-Templates in der Benutzeroberfläche –, bietet jedoch gleichzeitig die Chance, durch hybride Ansätze eine technologische

Vorreiterrolle einzunehmen.

Darüber hinaus beleuchtet der Bericht die kommerziellen Perspektiven des "Sear Drawbridge Proxy" als veräußerbares Produkt unter Berücksichtigung der GNU General Public License (GPL) und entwirft eine Roadmap für die Integration der komplexen "Measuring"-Algorithmen und der "Sparky"-Visualisierung. Ziel ist es, die Lücke zwischen der visionären "Nullstellen-Schablonen-Detektion" [1] und der pragmatischen Realität einer gehosteten WordPress.com-Business-Umgebung zu schließen.

2. Das "Sear Drawbridge Proxy" Template-Paradoxon: Eine Tiefenanalyse der WordPress-Architekturkonflikte

Die Beobachtung, dass das Template "Sear Drawbridge Proxy" trotz korrektem Upload der Datei im WordPress-Backend nicht zur Auswahl steht, ist symptomatisch für den aktuellen Konflikt zwischen Legacy-Strukturen und moderner Block-Architektur. Um dieses Problem nachhaltig zu lösen, ist ein tiefes Verständnis der Mechanismen erforderlich, wie WordPress Templates registriert, cached und im Kontext von Block-Themes (wie dem verwendeten Twenty Twenty-Four) priorisiert.

2.1. Architektonische Divergenz: PHP-Templates vs. Block-Templates

Historisch betrachtet, basierte die Template-Hierarchie von WordPress fast ausschließlich auf PHP-Dateien. Ein Theme bestand aus einer Kette von Fallback-Dateien (z.B. single.php, page.php, index.php), die von der WordPress Core Engine basierend auf der angeforderten URL (Query Var) aufgerufen wurden. Custom Page Templates wurden durch einen speziellen PHP-Kommentar-Header identifiziert:

PHP

```php
<?php
/*
Template Name: Sear Drawbridge Proxy
*/
?>
```

In der Ära der klassischen Themes scannte WordPress bei jedem Aufruf des Editors (oder beim Aktualisieren der Theme-Daten) das Verzeichnis rekursiv, extrahierte diese Header und füllte das "Template"-Dropdown in den Seitenattributen.[2]

Mit der Einführung von Full-Site-Editing (FSE) und Block-Themes hat sich dieses Paradigma verschoben. Themes wie Twenty Twenty-Four definieren ihre Struktur primär über HTML-Dateien im Ordner /templates und Konfigurationen in der theme.json. WordPress bevorzugt nun diese HTML-Templates, da sie im Site Editor visuell bearbeitet werden können. PHP-Templates werden zwar weiterhin aus Gründen der Abwärtskompatibilität unterstützt, jedoch behandelt der neue Block-Editor (Gutenberg) sie als Bürger zweiter Klasse.

Das spezifische Problem hier resultiert aus einer Kombination von drei Faktoren, die in der modernen WordPress-Entwicklung oft übersehen werden:

1. **Hierarchische Verdrängung:** Wenn ein Block-Theme eine page.html im /templates-Ordner besitzt, wird diese oft priorisiert, bevor WordPress überhaupt prüft, ob ein benutzerdefiniertes PHP-Template zugewiesen wurde, es sei denn, die Zuweisung ist in der Datenbank explizit und korrekt verankert.[3]
2. **Editor-Restriktionen:** Der neue Site Editor ist darauf ausgelegt, Block-Templates zu bearbeiten. Da PHP-Templates serverseitigen Code enthalten, der nicht live im Editor gerendert werden kann (ohne komplexe Server-Side-Rendering-Callbacks), blendet die Benutzeroberfläche in Versionen ab WordPress 6.5+ die Auswahlmöglichkeit für PHP-Templates oft aus, um "Verwirrung" beim Nutzer zu vermeiden.[4]
3. **Caching-Latenz:** Auf Managed-Hosting-Plattformen wie WordPress.com Business ist das Caching weitaus aggressiver als auf Standard-Shared-Hosting.

2.2. Die Rolle des Transienten-Caches und des Object-Cache

Ein oft unterschätzter Faktor bei der Template-Registrierung ist der sogenannte "Theme Cache". WordPress liest das Dateisystem nicht bei jedem Seitenaufruf aus – dies wäre performancetechnisch fatal. Stattdessen wird die Liste der verfügbaren Templates (Dateiname + Header-Name) in einem transienten Eintrag in der Datenbank (wp_options Tabelle) oder im Object Cache (Redis/Memcached) gespeichert.

Wenn eine neue Datei wie sear-drawbridge.php via SFTP oder Dateimanager hochgeladen wird, "weiß" die Datenbank noch nichts von ihrer Existenz. Normalerweise triggert ein Wechsel des Themes oder das Speichern der Theme-Einstellungen einen "Cache Flush". Auf der WordPress.com Business-Plattform sind diese Caches jedoch oft persistent und mehrschichtig [5]:

- **Object Cache:** Speichert Datenbankabfragen. Die Abfrage "Welche Templates gibt es?" wird hier oft minuten- oder stundenlang zwischengespeichert.
- **Edge Cache (CDN):** Speichert die fertige HTML-Ausgabe des Admin-Bereichs, was dazu führen kann, dass man eine veraltete Version des Editors sieht.

Das Dokument [5] bestätigt, dass auf dem Business-Plan ein manuelles Eingreifen erforderlich sein kann. Die bloße Existenz der Datei garantiert keine Sichtbarkeit im UI.

2.3. Lösungsweg: Erzwingung der Template-Erkennung und Umgehung der UI-Limitierung

Basierend auf dieser Analyse und den Erkenntnissen aus den Support-Foren [4] sowie der Dokumentation zur FSE-Kompatibilität [3], wird folgende detaillierte Prozedur zur Behebung des Problems empfohlen. Diese Schritte sind sequenziell auszuführen, um jede Ebene des Problems zu adressieren.

Schritt 1: Validierung der Datei-Integrität

Es muss sichergestellt werden, dass die Datei sear-drawbridge.php nicht nur physisch vorhanden ist, sondern auch syntaktisch korrekt beginnt. Ein fehlendes Leerzeichen oder ein falscher Zeilenumbruch im Header kann die Erkennung verhindern.

Der korrekte Inhalt muss zwingend so aussehen:

PHP

```php
<?php
/*
Template Name: Sear Drawbridge Proxy
*/

// WICHTIG: In Block-Themes fehlt oft der klassische Header-Hook.
// Wir müssen sicherstellen, dass wp_head() gefeuert wird, sonst laden CSS/JS nicht.
if (! defined( 'ABSPATH' ) ) {
	exit; // Exit if accessed directly.
}

get_header();
?>

<div class="itheereum-app-wrapper" style="min-height: 80vh; position: relative;">
</div>

<?php
get_footer();
?>
```

Hinweis: Die Datei muss direkt im Root-Verzeichnis /wp-content/themes/itheereum-child/ liegen. Unterordner wie /templates/ oder /inc/ werden von der klassischen Template-Scan-Logik oft ignoriert, besonders in Child-Themes.[7]

Schritt 2: Der "Hard Flush" auf WordPress.com

Da wir es mit einem Business-Plan zu tun haben, reicht ein einfaches Browser-Refresh nicht aus.

1. Navigieren Sie im WordPress-Dashboard zu **Einstellungen** > **Hosting-Konfiguration** (oder nutzen Sie die Tastenkombination Strg+K und suchen Sie nach "Cache").
2. Betätigen Sie den Button **"Alle Caches leeren"** (Clear all caches). Dies ist entscheidend, um den Object Cache zu bereinigen.[5]
3. **Der Theme-Switch-Trick:** Um WordPress zu zwingen, das Verzeichnis neu zu scannen, aktivieren Sie kurzzeitig das Eltern-Theme (Twenty Twenty-Four) und dann sofort wieder das Child-Theme. Dieser Prozess löscht intern die theme_roots Transiente in der Datenbank und baut die Liste der Templates neu auf.[9]

Schritt 3: Umgehung des Block-Editor-Bugs via "Quick Edit"

Wie in den Snippets [4] und [6] dokumentiert, gibt es einen spezifischen Bug in WordPress 6.5+, bei dem das Template-Dropdown in der Seitenleiste des Block-Editors verschwindet, wenn die Seite bereits einem FSE-Template zugewiesen ist.

1. Gehen Sie zur Übersicht **Seiten** > **Alle Seiten**.
2. Suchen Sie die Seite, auf der die App laufen soll.
3. Klicken Sie auf **Quick Edit** (Schnellbearbeitung).
4. Hier wird das "Template"-Dropdown oft noch angezeigt, da Quick Edit auf einer älteren Code-Basis beruht als der Block-Editor. Wählen Sie hier "Sear Drawbridge Proxy" aus und speichern Sie.

3. Die Robustheit des Child-Themes: Beständigkeit gegen "Digitale Entropie"

Die Sorge um die Beständigkeit des Child-Themes ("Ist das Child-Theme beständig gegen Updates?") ist angesichts der komplexen "Sear"-Architektur mit ihren "Weaving"-

Animationen und Canvas-Overlays [1] absolut berechtigt. Während Child-Themes theoretisch updatesicher sind, zeigt die Praxis Nuancen, die für die langfristige Stabilität der "Itheereum"-Plattform entscheidend sind.

3.1. Strukturelle Integrität vs. DOM-Drift

Ein Child-Theme schützt PHP-Dateien und die style.css vor dem Überschreiben bei einem Update des Eltern-Themes.[10] Das ist die Basis. Das größere Risiko, das in der Dokumentation [1] als "degenerative Repetition" angedeutet wird, ist jedoch der **DOM-Drift**.

Wenn das Eltern-Theme (Twenty Twenty-Four) aktualisiert wird, könnten sich die HTML-Klassen oder die Schachtelungstiefe der Container ändern (z.B. wechselt ein Wrapper von zu <main> oder eine Klasse .wp-block-group wird zu .wp-block-group-is-layout-constrained). Da die sear-style.css [1] auf präzise CSS-Selektoren angewiesen ist, um die "orthogonal korrekten" Animationen und den "Quantum-Glimmer" darzustellen, könnte ein Update des Eltern-Themes das Layout brechen, ohne eine einzige Datei im Child-Theme zu löschen.

Empfehlung zur Härtung:
Um maximale Beständigkeit zu erreichen, sollte das Template sear-drawbridge.php so autark wie möglich agieren. Anstatt sich auf die Header- und Footer-Parts des Eltern-Themes zu verlassen (die sich ändern können), sollte das Template idealerweise eine eigene, minimalistische Struktur definieren, die nur die absolut notwendigen WordPress-Hooks (wp_head, wp_footer) lädt. Dies isoliert die "Sear"-App in einer stabilen "Kapsel" innerhalb der Website.

3.2. Die Gefahr der theme.json Vererbung

In Block-Themes erbt das Child-Theme die Einstellungen der theme.json des Eltern-Themes (Farben, Schriftgrößen, Layout-Breiten).10 Ein Update des Eltern-Themes kann diese Werte ändern. Wenn Ihre "Sear"-App auf var(--wp--preset--color--base) zugreift und das Eltern-Theme diese Variable ändert, ändern sich Ihre Farben.

Die in 1 definierte Strategie, eigene CSS-Variablen (z.B. --onyx-black, --neon-purple) im Root der sear-style.css zu definieren, ist hierfür der perfekte Schutzmechanismus. Sie entkoppelt das Design der App vollständig von den Launen des Theme-Updates.

4. Kommerzialisierung und Lizenzierung: Das "Sear"-System als Produkt

Die Frage "Kann ich es als Produkt vermarkten?" berührt fundamentale Aspekte der Open-Source-Ökonomie. Die Antwort ist ein klares Ja, doch der Weg dorthin erfordert eine differenzierte Strategie, die die GPL-Lizenzbedingungen respektiert und gleichzeitig den proprietären Wert der "Itheereum"-Technologie schützt.

4.1. Rechtlicher Rahmen: Die GPL und ihre Grenzen

WordPress und alle davon abgeleiteten Werke (Derivative Works), zu denen technisch gesehen jedes Theme und Child-Theme gehört, müssen unter der GNU General Public License (GPL) veröffentlicht werden.12
Das bedeutet konkret:

- Sie dürfen das Theme verkaufen (auch für hohe Beträge).
- Sie können dem Käufer jedoch nicht verbieten, den Code zu kopieren, zu verändern oder weiterzuverschenken.
- Der PHP-Code des "Sear Drawbridge Proxy" Templates ist "infiziert" von der GPL, da er WordPress-Funktionen wie get_header() aufruft.

4.2. Das "Split License" Modell und Asset-Schutz

Es gibt jedoch eine etablierte Interpretation (u.a. von Envato und anderen Marktplätzen genutzt), das "Split License" Modell.[13] Hierbei wird argumentiert, dass Assets wie CSS, JavaScript, Bilder (Emojis) und Design-Patterns nicht zwingend der GPL unterliegen

müssen, wenn sie als eigenständige Werke betrachtet werden können.

- Die **CSS-Dateien** (sear-style.css mit den "Weaving"-Animationen) und die **JavaScript-Module** (sear-measuring.js, sparky-loader.js) könnten theoretisch unter einer proprietären Lizenz stehen, solange sie nicht direkt PHP-Funktionen von WordPress enthalten.
- Die **Emojis** aus [1] und die **Musikstücke** (MP3/WAV) sind urheberrechtlich geschützte Assets ("Creative Works"), die Sie unter Ihren eigenen Bedingungen lizenzieren können.

4.3. Strategie: "Service-as-a-Product" und Tokenisierung

Der eigentliche Wert von "Sear" liegt nicht im Child-Theme-Code (der kopiert werden kann), sondern in der **Intelligenz der "Defender OHM" Cloud-Architektur** und den Updates.

Ich empfehle folgende Vermarktungsstrategie, die sich an erfolgreichen Premium-Plugins orientiert (z.B. Elementor Pro, WP Rocket):

1. **Das Theme als "Schlüssel":** Vermarkten Sie das Child-Theme als "Interface-Client". Der Code ist GPL, aber um die "Measuring"-Funktion (die Parallel-Suche via Google Grounding und DDG) nutzen zu können, benötigt der Nutzer einen **API-Key**.
2. **SaaS-Anbindung:** Dieser API-Key authentifiziert den Nutzer gegenüber Ihrer "Defender OHM" Infrastruktur.[1] Ohne validen Key läuft die App im "Sandkasten-Modus" (lokale Suche), mit Key schaltet sich die volle "Cybercity"-Power frei.
3. **Verkaufsplattform:** Nutzen Sie für den Vertrieb Lösungen wie **WooCommerce** (mit dem Plugin "WooCommerce Software License") oder **Easy Digital Downloads**.[14] Für das Packaging des Themes empfiehlt sich ein Tool wie **SitePresser** [15], das hilft, das Theme sauber zu bündeln und unerwünschte Entwickler-Dateien auszuschließen.

5. Technische Implementierung: Integration des "Arrow" Dropdowns und Media-Players

Die Platzierung und technische Realisierung des "Arrow"-Dropdowns [1] erfordert Präzision, um die Integrität des Designs und die Funktionalität der asynchronen Module ("Sparky", "Measuring") zu gewährleisten.

5.1. Die Lokalisierung: Wo gehört der Code hin?

Die Frage "Wo genau füge ich den HTML-Code für das 'Arrow'-Dropdown ein?" hat zwei Antworten, abhängig von der gewählten Integrationsmethode (FSE vs. Hybrid).

Option A: Integration in die itheereum-sear-app.html (Empfohlen für Kapselung)

Da die gesamte App modular aufgebaut ist [1], ist der sauberste Ort für den Arrow-Code direkt innerhalb der Strukturdatei itheereum-sear-app.html. Dies stellt sicher, dass der Arrow Teil des "Sear"-Ökosystems bleibt und Zugriff auf den lokalen Scope der App hat (z.B. für die Steuerung des Media-Players).

Platzierung im Code:
Der Arrow sollte semantisch nach dem Suchfeld (.sear-mount), aber vor den Ergebnissen und Tabs platziert werden. Dies schafft eine logische Hierarchie: Suche -> Steuerung/Info (Arrow) -> Ergebnisse.

HTML

```
<form class="sear-mount" id="sear-form">...</form>
```

```
<div class="arrow-wrapper-container" style="text-align: right; margin-top: 1rem; position: relative;">

    <div class="finding-summary-text" style="font-size: 0.8rem; color: var(--crystal-cyan-light); margin-bottom: 0.2rem;">
        <span class="date">05.11.2025</span> | <a href="#" class="source-link">Itheereum Medley</a>
    </div>

    <button id="sear-arrow-trigger" class="arrow-button" aria-expanded="false" onclick="toggleArrowSandbox()">
        Arrow ▼
    </button>

    <div id="arrow-sandbox-panel" class="transparency-toggle" style="display: none;">
        <div class="weaving-player-placeholder">
            <h4>Weaving Media Player</h4>
            </div>
    </div>
</div>
<div class="spacer-tabs">...</div>
```

Option B: Injektion via functions.php oder Custom Block (Für globale Sichtbarkeit)

Wenn der Arrow, wie in 1 angedeutet, "oben rechts... in der Titelleiste der Homepage" erscheinen soll, also außerhalb des Content-Bereichs der Sear-App, müssen wir aus dem App-Container ausbrechen.
In einem FSE-Theme wie Twenty Twenty-Four geht dies am besten über den Site Editor:

1. Öffnen Sie **Design** > **Editor** > **Vorlagen** > **Header**.
2. Fügen Sie einen **"Individuelles HTML"**-Block (Custom HTML) hinzu.[16]
3. Fügen Sie dort den Arrow-HTML-Code ein.
4. Damit die Funktionalität erhalten bleibt, muss das JavaScript (sear-measuring.js) so angepasst werden, dass es auch Elemente außerhalb des Main-Containers findet (Verwendung von document.getElementById statt relativer Selektoren).

5.2. Design-Spezifikationen für den "Arrow" (CSS & Animation)

Um die Anforderung "Dunkellicht-Spektrum, geschwungene bis scharfe Form" und "sich bewegender Leucht-Effekt" [1] zu erfüllen, muss die sear-style.css erweitert werden. Wir nutzen hierfür cubic-bezier Kurven für die "orthogonale Korrektheit" der Bewegung.

CSS

```
/* Arrow Button Styling */
.arrow-button {
    background: linear-gradient(135deg, var(--galaxy-blue-dark) 0%, var(--onyx-black) 100%);
    border: 1px solid var(--neon-purple);
    color: var(--crystal-cyan-light);
    padding: 0.5rem 1.5rem;
    /* Geschwungene Form: Oben scharf, unten rund */
    border-radius: 4px 4px 20px 20px;
    cursor: pointer;
    font-family: var(--font-main);
    font-weight: 600;
    text-shadow: 0 0 8px var(--neon-cyan);
    position: relative;
    overflow: hidden;
    transition: all 0.4s cubic-bezier(0.175, 0.885, 0.32, 1.275); /* Orthogonale Federung */
}

/* Leucht-Effekt (Weaving Glow) */
.arrow-button::after {
    content: '';
    position: absolute;
    top: -50%;
    left: -50%;
    width: 200%;
```

```
    height: 200%;
    background: radial-gradient(circle, rgba(217, 70, 239, 0.2) 0%, transparent 70%);
    transform: scale(0);
    transition: transform 0.6s ease-out;
}

.arrow-button:hover {
    border-color: var(--neon-cyan);
    box-shadow: 0 0 20px rgba(34, 211, 238, 0.4);
    transform: translateY(4px) scale(1.02); /* Haptisches Feedback */
}

.arrow-button:hover::after {
    transform: scale(1);
}

/* Sandbox Panel (Fenster-in-Fenster) */
#arrow-sandbox-panel {
    margin-top: 10px;
    background: rgba(26, 26, 31, 0.85); /* Transparenz */
    border: 1px solid var(--neon-cyan);
    border-radius: 12px;
    padding: 1rem;
    /* Quantum Blur Effekt für Hintergrundunschärfe */
    backdrop-filter: blur(12px);
    -webkit-backdrop-filter: blur(12px);
    box-shadow: 0 10px 30px rgba(0,0,0,0.5);
    animation: panel-open 0.4s ease-out forwards;
}

@keyframes panel-open {
    from { opacity: 0; transform: translateY(-10px) scale(0.95); }
    to { opacity: 1; transform: translateY(0) scale(1); }
}
```

5.3. Der "Weaving" Media-Player: Integration und Formate

Der Media-Player, der im "Arrow"-Dropdown leben soll, ist mehr als ein einfaches <audio>-Tag. Er ist ein Kernmodul der MultiApp und soll "Canvas-Coding-Kontext" erfassen.[1]

Technische Umsetzung:

1. **Audio-Engine:** Nutzen Sie die Web Audio API anstelle des Standard-HTML5-Players. Dies ermöglicht die Analyse der Frequenzen in Echtzeit ("Measuring") und die Visualisierung auf dem Canvas.
2. **Format-Support:** Während MP3 und WAV nativ unterstützt werden, erfordert die Erwähnung von .swf (Flash) in [1] und der Verweis auf den "Open Flash Fork" [1] eine spezielle Behandlung. Da moderne Browser Flash nicht mehr unterstützen, müsste hier ein Emulator wie **Ruffle** (WebAssembly-basiert) eingebunden werden, um die .swf Dateien im Canvas zu rendern.
3. **Playlists:** Die "Top 3 Playlists" können als JSON-Array in der sear-measuring.js hinterlegt werden und dynamisch in das Arrow-Dropdown injiziert werden.

6. Die "Measuring"-Logik und Server-Performance

Ein kritischer Aspekt, der in der Strategie nicht übersehen werden darf, ist die Server-Last, die durch das "Measuring"-Protokoll erzeugt wird. Das Dokument [1] beschreibt eine simulierte Parallel-Abfrage (Aktion A + B). In der produktiven Umgebung ("Cybercity") wird dies reale API-Calls an Google und DuckDuckGo bedeuten.

6.1. Asynchrone Verarbeitung und "Defender OHM"

Das Child-Theme darf diese Abfragen **nicht** synchron im PHP-Prozess ausführen (das würde die Seite extrem verlangsamen). Stattdessen muss das Frontend (sear-measuring.js) die Anfrage an einen Endpunkt des "Defender OHM" (Google Cloud Function) senden. Dieser "Proxy" führt die parallelen Suchen durch, wendet den "Quer-

Vergleich"-Algorithmus an und sendet nur das gefilterte JSON-Ergebnis zurück an das WordPress-Frontend.

Dies bestätigt erneut die Notwendigkeit, das Template sear-drawbridge.php als reinen Container zu betrachten. Die Logik liegt nicht in WordPress, sondern im JavaScript und in der Cloud. Dies macht das System extrem robust gegen WordPress-Updates, da die Kernfunktionalität entkoppelt ist.

7. Fazit und konsolidierte Roadmap

Die Implementierung der "Itheereum Sear App" auf der bestehenden WordPress-Infrastruktur ist ein ambitioniertes, aber technisch machbares Unterfangen. Die aktuellen Probleme mit dem Template "Sear Drawbridge Proxy" sind keine Sackgasse, sondern ein typisches Symptom des Übergangs zu Full-Site-Editing, das durch präzise Eingriffe (Datei-Verschiebung, Caching-Flush, Quick-Edit-Zuweisung) behoben werden kann.

Das Projekt bietet durch seine einzigartige Kombination aus "Weaving"-Ästhetik (CSS/Canvas), "Measuring"-Philosophie (Wahrheitsfindung durch Quer-Vergleich) und modulare Architektur ein erhebliches Marktpotenzial. Die Kommerzialisierung sollte sich auf den "Service"-Aspekt (API-Zugang, Updates, Cloud-Intelligence) konzentrieren, während der Client-Code (das Theme) als Open-Source-Trägermedium dient.

Handlungsanweisungen (Priorisiert):

1. **Template-Fix:** Verschieben der sear-drawbridge.php in den Root-Ordner, Ergänzung des PHP-Headers und Ausführung des "Cache Flush" auf WordPress.com.
2. **Arrow-Integration:** Einbau des HTML-Containers in itheereum-sear-app.html und Ergänzung der CSS-Klassen in sear-style.css für die "Weaving"-Effekte.
3. **Media-Player-Prototyping:** Entwicklung der JS-Steuerung für das Arrow-Panel, initial mit Standard-Audio, später erweitert um Canvas-Visualisierung.
4. **Produkt-Packaging:** Vorbereitung des Child-Themes als zip-Datei, Bereinigung von hardcodierten Pfaden und Einrichtung einer Lizenzierungs-Schnittstelle (Vorbereitung für WooCommerce/SitePresser).

Mit diesen Schritten transformieren Sie den "Sandkasten" in eine robuste "Starfortress"

im digitalen Raum.

Referenzen

1. 03 Strategie zur Implementierung der Sear-App, Modulare Architektur, Measuring-Protokoll und Sparky-Weaving-Visualisierung.pdf
2. Naming convention for WordPress child theme - Stack Overflow, Zugriff am November 18, 2025, https://stackoverflow.com/questions/22450058/naming-convention-for-wordpress-child-theme
3. How to use PHP templates in block themes - Full Site Editing, Zugriff am November 18, 2025, https://fullsiteediting.com/lessons/how-to-use-php-templates-in-block-themes/
4. Page Templates no longer Appears after WordPress 6.5 – Block Editor, Zugriff am November 18, 2025, https://wordpress.org/support/topic/page-templates-no-longer-appears-after-wordpress-6-5-block-editor/
5. Clear your site's cache - WordPress.com, Zugriff am November 18, 2025, https://wordpress.com/support/clear-your-sites-cache/
6. Difficulty changing templates in page attributes - WordPress.org, Zugriff am November 18, 2025, https://wordpress.org/support/topic/difficulty-changing-templates-in-page-attributes/
7. PHP templates don't show as template option to select : r/Wordpress - Reddit, Zugriff am November 18, 2025, https://www.reddit.com/r/Wordpress/comments/1d9pu0u/php_templates_dont_show_as_template_option_to/
8. How to Clear WordPress Cache: Quick & Easy Tips - OneNine, Zugriff am November 18, 2025, https://onenine.com/how-to-clear-wordpress-cache/
9. Page template attribute missing - Support - Themeco Forum, Zugriff am November 18, 2025, https://theme.co/forum/t/page-template-attribute-missing/16336
10. Create a child theme – WordPress.com Support, Zugriff am November 18, 2025, https://wordpress.com/support/themes/child-themes/
11. Child themes - Learn WordPress, Zugriff am November 18, 2025, https://learn.wordpress.org/lesson/child-themes/
12. Selling a child-theme of twenty-thirteen is legal? [closed] - WordPress Stack Exchange, Zugriff am November 18, 2025, https://wordpress.stackexchange.com/questions/160070/selling-a-child-theme-of-twenty-thirteen-is-legal
13. Selling your theme : r/Wordpress - Reddit, Zugriff am November 18, 2025, https://www.reddit.com/r/Wordpress/comments/1fw5x2n/selling_your_theme/

14. How To Sell WordPress Themes - A Marketing Guide For Developers - inflowlabs.com, Zugriff am November 18, 2025, https://inflowlabs.com/blog/how-to-sell-wordpress-themes-a-marketing-guide-for-developers/
15. How to Build & Sell Child Themes for Divi or Elementor - SitePresser, Zugriff am November 18, 2025, https://sitepresser.io/how-to-build-sell-child-themes-for-divi-or-elementor/
16. Zugriff am November 18, 2025, https://wordpress.com/support/wordpress-editor/blocks/custom-html-block/#:~:text=To%20add%20the%20Custom%20HTML,page%2C%20post%2C%20or%20template.&text=Using%20your%20keyboard%2C%20you%20can,a%20new%20Custom%20HTML%20block.
17. Custom HTML block – WordPress.com Support, Zugriff am November 18, 2025, https://wordpress.com/support/wordpress-editor/blocks/custom-html-block/