# Уніфікована Архітектура Адаптивних Інтерфейсів 2025: Стандарти Web та Android для Систем Штучного Інтелекту

## 1. Стратегічний Контекст: Парадигма Інтринсичної Адаптації

Наприкінці 2025 року ландшафт розробки користувацьких інтерфейсів зазнав фундаментальної трансформації, відійшовши від класичного розділення на "мобільний дизайн" та "десктопний дизайн". Ми увійшли в еру **інтринсичного (внутрішнього) дизайну**, де компоненти інтерфейсу володіють власною логікою адаптації до контейнера, в якому вони знаходяться, незалежно від глобального розміру вікна перегляду (viewport). Для системи штучного інтелекту, яка має завдання автоматично генерувати код для веб-платформ та мобільних додатків Android, це означає перехід від жорстких медіа-запитів до гнучких алгоритмічних правил.

Цей звіт є вичерпним керівництвом, що синтезує найновіші стандарти станом на кінець 2025 року. Він об’єднує специфікацію W3C Design Tokens (версія 2025.10), новітні можливості CSS (Container Queries, View Transitions Level 2) та архітектуру Android 16 (API Level 36) з використанням Material Design 3 Expressive. Мета документа — надати структуровану логічну базу для побудови систем, які є "ідеально адаптованими" за своєю природою, а не через набір милиць та винятків.

### 1.1. Еволюція від Responsive до Adaptive

Традиційний "Responsive Web Design" (RWD), що домінував останнє десятиліття, покладався на глобальні зміни макета при досягненні певних точок зупинки (breakpoints) ширини екрана. У 2025 році цей підхід вважається застарілим для складних інтерфейсів. На зміну прийшов "Adaptive Design", який передбачає зміну не лише розташування елементів, але й самої структури та поведінки компонентів.1

Аналіз показує, що критична відмінність полягає в контексті:

* **Responsive (Реагуючий):** "Якщо екран менше 600px, склади колонки в стопку".
* **Adaptive (Адаптивний):** "Якщо контейнер занадто вузький для відображення детального контенту, заміни структуру списку на навігаційну панель, що веде до окремого екрану".

Ця різниця є фундаментальною для Android 16 та сучасного CSS. ШІ-архітектор повинен розуміти, що "мобільна версія" більше не є окремим сутнісним станом, а є однією з можливих конфігурацій контейнера, яка може виникнути навіть на великому екрані (наприклад, у режимі розділеного екрана або у плаваючому вікні).2

## 2. Фундамент Уніфікації: W3C Design Tokens (Специфікація 2025.10)

Аби забезпечити синхронізацію між веб-сайтом та Android-додатком, необхідно використовувати єдине джерело істини. У жовтні 2025 року W3C Design Tokens Community Group випустила першу стабільну версію специфікації Design Tokens Format Module.3 Це "ДНК" дизайн-системи, яку ШІ повинен використовувати для генерації коду.

### 2.1. Архітектура Токенів та Формат JSON

Специфікація вимагає використання формату JSON, де кожен токен є об'єктом з обов'язковими властивостями $value та $type. Це дозволяє інструментам автоматично транслювати значення у CSS Custom Properties (для Web) та об'єкти Kotlin (для Android Jetpack Compose).4

#### 2.1.1. Ієрархія Токенів

Для побудови масштабованої системи ШІ повинен організувати токени у трирівневу структуру 5:

| **Рівень** | **Опис** | **Приклад Назви** | **Значення (Web / Android)** |
| --- | --- | --- | --- |
| **Primitive (Reference)** | Базові значення без контексту (палітра, шкала розмірів). | color.blue.500 | #0066FF / Color(0xFF0066FF) |
| **Semantic (System)** | Значення, прив'язані до контексту використання (роль у системі). | color.background.primary | {color.blue.500} (Light) / {color.blue.300} (Dark) |
| **Component** | Значення, специфічні для конкретного компонента. | button.bg.hover | {color.background.primary} |

#### 2.1.2. Композитні Токени (Composite Tokens)

Нововведенням специфікації 2025 року є композитні токени, які дозволяють групувати пов'язані властивості. Це критично для таких елементів, як типографіка та тіні, що не можуть бути описані одним значенням.

ШІ повинен парсити композитні токени як єдині об'єкти стилю:

* **Тінь (Shadow):** Об'єкт, що містить offsetX, offsetY, blur, spread та color.4
* **Типографіка (Typography):** Об'єкт, що містить fontFamily, fontSize, fontWeight, lineHeight.

**Приклад JSON для ШІ-парсингу:**

JSON

{  
 "elevation": {  
 "card-default": {  
 "$type": "shadow",  
 "$value": {  
 "color": "{color.shadow.base}",  
 "offsetX": "0px",  
 "offsetY": "4px",  
 "blur": "8px",  
 "spread": "0px"  
 }  
 }  
 }  
}

### 2.2. Правила Іменування та Аліасинг

Для забезпечення сумісності між платформами, специфікація накладає суворі обмеження на іменування. Заборонено використовувати символи {, }, . у назвах токенів, оскільки вони використовуються для синтаксису посилань (aliasing). ШІ повинен використовувати крапкову нотацію лише для позначення вкладеності груп у JSON.4

**Логіка для ШІ:** При генерації CSS змінних, крапки у шляху JSON замінюються на дефіси (наприклад, color.blue.500 стає --color-blue-500). При генерації Kotlin-коду, вони перетворюються на об'єкти або властивості (наприклад, AppTheme.color.blue500).

## 3. Архітектура Web 2025: Інтринсична Верстка та Container Queries

Керівництво з веб-розробки на кінець 2025 року диктує відмову від використання vw (viewport width) та глобальних медіа-запитів для компонентної верстки. Основним інструментом стають **CSS Container Queries**.7

### 3.1. Container Queries: Математика Контексту

На відміну від медіа-запитів, запити контейнера дозволяють компоненту знати, скільки місця йому виділено його батьківським елементом. Це вирішує проблему повторного використання компонентів: картка товару виглядатиме ідеально як у широкій сітці каталогу, так і у вузькому віджеті "схожі товари", використовуючи один і той самий CSS код.

#### 3.1.1. Одиниці Виміру Контейнера

ШІ повинен використовувати нові одиниці виміру для визначення розмірів внутрішніх елементів компонента:

* **cqw (Container Query Width):** 1% від ширини контейнера.
* **cqi (Container Query Inline):** 1% від логічної ширини (inline-size). Це стандарт для підтримки міжнародних інтерфейсів (RTL/LTR).
* **cqmin / cqmax:** Менше або більше з двох вимірів контейнера.7

#### 3.1.2. Впровадження та Синтаксис

Для активації цієї логіки, ШІ повинен згенерувати CSS, що явно визначає контекст контейнера:

CSS

/\* Визначення контейнера \*/  
.product-card-wrapper {  
 container-type: inline-size;  
 container-name: card-context;  
}  
  
/\* Використання запиту \*/  
@container card-context (min-width: 400px) {  
 .product-card {  
 display: grid;  
 grid-template-columns: 1fr 2fr; /\* Перехід в горизонтальний режим \*/  
 gap: 1rem;  
 }  
   
 .product-title {  
 font-size: clamp(1rem, 5cqi, 1.5rem); /\* Адаптивний шрифт на основі контейнера \*/  
 }  
}

### 3.2. Style Queries: Тематизація без Класів

У 2025 році CSS дозволяє компонентам адаптуватися не лише до розміру, але й до обчислених стилів контейнера (Style Queries). Це особливо актуально для реалізації темної теми або режимів високого контрасту без необхідності прокидати додаткові CSS-класи через JavaScript.7

Приклад логіки:

ШІ може налаштувати компонент так, щоб він автоматично змінював колір тексту, якщо його контейнер має певний CSS Custom Property:

CSS

@container style(--theme: dark) {  
 .card-content {  
 background-color: var(--surface-dark);  
 color: var(--on-surface-dark);  
 }  
}

Це дозволяє створювати повністю ізольовані, портативні UI-компоненти, які "знають", як поводитися в будь-якому контексті.

## 4. Розширена Взаємодія Web: Анімації та Core Web Vitals

"Ідеальна адаптація" включає в себе не лише статичну верстку, але й плавність переходів між станами. Стандарт 2025 року вимагає використання нативних браузерних API для анімації, що мінімізує навантаження на головний потік виконання JavaScript.

### 4.1. View Transitions API (Level 2) для MPA

Межа між Single Page Applications (SPA) та Multi-Page Applications (MPA) фактично стерта завдяки View Transitions API. Ця технологія дозволяє анімувати переходи між *різними* HTML-документами при повному перезавантаженні сторінки, забезпечуючи досвід нативного додатку.9

#### 4.1.1. Декларативна Навігація

ШІ повинен додати наступне правило у глобальний CSS, щоб активувати плавні переходи за замовчуванням (cross-fade):

CSS

@view-transition {  
 navigation: auto;  
}

#### 4.1.2. Персистентність Елементів

Для створення ефекту "морфінгу" (коли зображення товару зі списку плавно розгортається на сторінці деталей), необхідно використовувати властивість view-transition-name.

* **Вимога:** Унікальність імені в межах поточної сторінки.
* **Алгоритм для ШІ:** При генерації списку елементів, ШІ повинен динамічно присвоювати view-transition-name (наприклад, product-image-ID123) як для мініатюри на сторінці лістингу, так і для основного зображення на сторінці продукту. Браузер автоматично інтерполює розмір та позицію між цими двома станами під час навігації.9

### 4.2. Scroll-Driven Animations

Для ефектів, що залежать від прокрутки (паралакс, індикатори читання, поява елементів), стандарт 2025 року забороняє використання JavaScript-слухачів подій scroll. Замість цього використовуються CSS Scroll Timelines.11

Переваги: Анімація виконується в окремому потоці композитора (Compositor Thread), що гарантує 60/120 FPS навіть якщо головний потік заблокований важкими обчисленнями.

Синтаксис:

CSS

.progress-bar {  
 animation: grow-progress linear;  
 animation-timeline: scroll(); /\* Прив'язка до скролу сторінки \*/  
}

### 4.3. Core Web Vitals 2025: INP як Головний Метрика

Оптимізація під Core Web Vitals у 2025 році зосереджена на метриці **Interaction to Next Paint (INP)**, яка повністю замінила First Input Delay (FID).13

**Порогові значення для 2025 року:**

| **Метрика** | **Добре (Good)** | **Потребує покращення** | **Погано (Poor)** |
| --- | --- | --- | --- |
| **INP** (Responsiveness) | ≤ 200 мс | 200–500 мс | > 500 мс |
| **LCP** (Loading Speed) | ≤ 2.5 с | 2.5–4.0 с | > 4.0 с |
| **CLS** (Visual Stability) | ≤ 0.1 | 0.1–0.25 | > 0.25 |

Стратегія для ШІ:

Для забезпечення низького INP, ШІ повинен структурувати JavaScript код так, щоб уникати довгих задач (Long Tasks > 50ms). Рекомендується використання scheduler.yield() або розбиття задач через setTimeout для звільнення головного потоку, що дозволяє браузеру реагувати на введення користувача миттєво.15

## 5. Нативна Архітектура Android 2025: Android 16 та Material 3 Expressive

Розробка додатків для Android на кінець 2025 року вимагає відповідності стандартам Android 16 (API Level 36) та використання оновленої дизайн-системи Material 3 Expressive. Основний фокус змістився на "емоційний UX" та обов'язкову адаптивність під різні форм-фактори (складні пристрої, планшети, десктопний режим Android).

### 5.1. Material 3 Expressive: Фізика та Емоції

Нова ітерація Material Design впроваджує більш "живу" фізику взаємодії.16 Стандартні анімації переходу замінюються на анімації на основі пружин (spring-based physics).

#### 5.1.1. Нові Компоненти

ШІ повинен використовувати нові компоненти з бібліотеки androidx.compose.material3:

* **Split Buttons:** Об'єднані кнопки для основної та додаткової дії.
* **Flexible Toolbars:** Панелі інструментів, що можуть змінювати своє положення та розмір залежно від контексту.
* **Loading Indicators:** Хвилеподібні індикатори прогресу замість кругових спінерів.16

#### 5.1.2. Haptic Feedback (Тактильний відгук)

Android 16 впроваджує нові API для тактильного відгуку, що дозволяють контролювати амплітуду та частоту вібрації. ШІ повинен використовувати HapticFeedbackConstants для підсилення візуальних ефектів "Expressive" дизайну, створюючи фізичне відчуття взаємодії з цифровими об'єктами.18

### 5.2. Edge-to-Edge Enforcement (Примусовий Повноекранний Режим)

Важливою зміною в Android 16 є скасування можливості відмовитися від режиму Edge-to-Edge. Атрибут windowOptOutEdgeToEdgeEnforcement застарів і ігнорується.19

**Імплікації для ШІ-генератора коду:**

* Додаток *завжди* малюється під системними панелями (status bar, navigation bar).
* ШІ повинен обов'язково впроваджувати обробку WindowInsets у Jetpack Compose, щоб контент не перекривався системними елементами.
* Використання Scaffold(contentWindowInsets = WindowInsets.systemBars) стає обов'язковим стандартом.

## 6. Адаптивні Макети Android: Стратегії Reflow та Levitate

Подібно до вебу, Android-розробка відмовляється від фіксованих орієнтацій екрана. Додатки повинні адаптуватися до змін розміру вікна в реальному часі (наприклад, при розгортанні складного смартфона).

### 6.1. Window Size Classes

Основою адаптації є класи розмірів вікна: **Compact, Medium, Expanded, Large, Extra-Large**.20 ШІ повинен використовувати функцію currentWindowAdaptiveInfo() для отримання поточного класу та перебудови інтерфейсу.

### 6.2. Канонічні Макети та Стратегії Адаптації

Бібліотека androidx.compose.material3.adaptive (версія 1.3+) пропонує готові рішення, які ШІ повинен імплементувати.21

#### 6.2.1. List-Detail Pane Scaffold

Це стандарт для додатків типу "пошта" або "месенджер".

* **Compact:** Показує список. При натисканні — перехід на деталі.
* **Expanded:** Показує список і деталі поруч.

#### 6.2.2. Стратегії Адаптації (Adaptive Strategies)

ШІ повинен розрізняти та застосовувати дві основні стратегії поведінки при зміні розміру вікна:

1. **Reflow (Перетікання):** Елементи змінюють своє розташування, заповнюючи доступний простір (наприклад, сітка з 2 колонок стає сіткою з 4 колонок). Це аналог Flexbox/Grid поведінки.
2. **Levitate (Левітація):** Нова стратегія 2025 року. При розширенні вікна, допоміжний контент (наприклад, панель редагування), який був прихований або знаходився в діалозі, "випливає" (levitates) і стає постійною бічною панеллю, не перериваючи контекст користувача.21

**Приклад коду Compose для ШІ:**

Kotlin

val navigator = rememberListDetailPaneScaffoldNavigator()  
ListDetailPaneScaffold(  
 directive = navigator.scaffoldDirective,  
 value = navigator.scaffoldValue,  
 listPane = { /\* List Content \*/ },  
 detailPane = { /\* Detail Content \*/ }  
)

### 6.3. Predictive Back Navigation

Підтримка предиктивного жесту "Назад" є обов'язковою в Android 16. Системні анімації (згортання додатку в іконку) увімкнені за замовчуванням. ШІ повинен замінити застарілі методи onBackPressed на OnBackInvokedCallback, щоб забезпечити коректну інтеграцію з системними жестами.19

## 7. Доступність та Інклюзивність (Accessibility)

На кінець 2025 року вимоги до доступності стали суворішими, зокрема через наближення European Accessibility Act (EAA).

### 7.1. WCAG 2.2 та Підготовка до 3.0

Хоча WCAG 3.0 (Project Silver) все ще знаходиться в стадії чернетки (Working Draft), стандартом де-факто і де-юре є WCAG 2.2.22

ШІ повинен забезпечити відповідність новим критеріям:

* **Focus Appearance (2.4.13):** Індикатор фокусу повинен мати достатній контраст (мінімум 3:1) і розмір.
* **Dragging Movements (2.5.7):** Всі дії, що вимагають перетягування, повинні мати альтернативу через прості кліки (single pointer).
* **Target Size (2.5.8):** Мінімальний розмір цілі натискання — 24x24 CSS пікселів (для Web) та 48x48 dp (для Android).

### 7.2. Android 16 Accessibility Features

Android 16 вводить функцію **Outline Text** для покращення читабельності. Текст отримує висококонтрастну обводку на рівні системи.

* **Вимога до ШІ:** При генерації UI компонентів, уникати жорсткого задавання фонів для тексту, які можуть конфліктувати з системною обводкою. Використовувати семантичні ролі Compose (Modifier.semantics) для коректної роботи TalkBack.24

## 8. Технічний Синтез: Інструкція для ШІ-Будівельника

Цей розділ містить конкретні правила для іншого ШІ, який буде генерувати код на основі цього звіту.

### 8.1. Правила Web-генерації

1. **Починати з Токенів:** Згенерувати tokens.json згідно зі специфікацією W3C 2025.10.
2. **CSS Архітектура:** Використовувати @layer для управління каскадом.
3. **Layout Logic:** Заборонити використання медіа-запитів для компонентів. Тільки container-type та @container.
4. **Interaction:** Для переходів між сторінками використовувати <link rel="expect"> та @view-transition.
5. **Performance:** Всі анімації скролу реалізовувати через animation-timeline.

### 8.2. Правила Android-генерації

1. **Theme Mapping:** Транслювати семантичні токени в MaterialTheme.colorScheme.
2. **Adaptive Root:** Кожен екран повинен бути обгорнутий в адаптивний Scaffold (наприклад, NavigationSuiteScaffold або ListDetailPaneScaffold).
3. **Window Insets:** Явно обробляти відступи через WindowInsets.
4. **Motion:** Використовувати пружинні анімації (spring()) для всіх змін стану layout.

### 8.3. Таблиця Відповідності Технологій

| **Функція** | **Web (2025 Implementation)** | **Android (Jetpack Compose 2025)** |
| --- | --- | --- |
| **Визначення Контексту** | CSS Container Queries (@container) | currentWindowAdaptiveInfo() |
| **Адаптація Макета** | CSS Grid / Flexbox Reflow | ListDetailPaneScaffold / SupportingPaneScaffold |
| **Анімація Переходів** | View Transitions API | Shared Element Transitions / Predictive Back |
| **Скрол-ефекти** | Scroll-driven Animations (CSS) | Modifier.nestedScroll / RenderEffect |
| **Колірна Схема** | CSS Variables (HCT / OKLCH) | Material You (Dynamic Color) |
| **Доступність** | ARIA Roles / WCAG 2.2 | Semantics Modifier / Android 16 A11y API |

## Висновок

Побудова ідеально адаптованого сайту та додатку у 2025 році вимагає відмови від застарілих практик фіксованої верстки на користь гнучких, контекстно-залежних систем. Використання W3C Design Tokens як спільної мови, Container Queries для веб-логіки та адаптивних бібліотек Material 3 для Android дозволяє створити продукт, який є не просто "сумісним" з мобільними пристроями, а органічно існує в будь-якому цифровому середовищі — від годинника до настільного монітора. Цей звіт надає повний набір інструментів та логічних правил для реалізації такої системи автономним агентом штучного інтелекту.

#### Джерела

1. Scaling Compose with Adaptive UIs, Testing, and Type-Safe Navigation, доступ отримано грудня 24, 2025, <https://sevenpeakssoftware.com/blog/scaling-compose-with-adaptive-uis-testing-and-type-safe-navigation>
2. Google I/O 2025: Build adaptive Android apps that shine across form factors, доступ отримано грудня 24, 2025, <https://android-developers.googleblog.com/2025/05/adaptiveapps-io25.html>
3. Design Tokens specification reaches first stable version - W3C, доступ отримано грудня 24, 2025, <https://www.w3.org/community/design-tokens/2025/10/28/design-tokens-specification-reaches-first-stable-version/>
4. Design Tokens Format Module 2025.10, доступ отримано грудня 24, 2025, <https://www.designtokens.org/tr/2025.10/format/>
5. Design Tokens & Theming: How to Build Scalable UI Systems in 2025 - Material UI, доступ отримано грудня 24, 2025, <https://materialui.co/blog/design-tokens-and-theming-scalable-ui-2025>
6. Design System Concepts and Establishing Basic Design Tokens, доступ отримано грудня 24, 2025, <https://www.designsystemscollective.com/1-design-system-concepts-and-establishing-basic-design-tokens-b1a974da0c16>
7. CSS 2025 Container queries and style queries in real projects | by Valentyn Yakymenko, доступ отримано грудня 24, 2025, <https://medium.com/@vyakymenko/css-2025-container-queries-and-style-queries-in-real-projects-c38af5a13aa2>
8. CSS techniques every developer should know in 2025 - DEV Community, доступ отримано грудня 24, 2025, <https://dev.to/dimeloper/css-techniques-every-developer-should-know-in-2025-30p9>
9. View Transition API - Web APIs | MDN, доступ отримано грудня 24, 2025, <https://developer.mozilla.org/en-US/docs/Web/API/View_Transition_API>
10. A beginner-friendly guide to view transitions in CSS - MDN Web Docs, доступ отримано грудня 24, 2025, <https://developer.mozilla.org/en-US/blog/view-transitions-beginner-guide/>
11. CSS scroll-driven animations - MDN Web Docs, доступ отримано грудня 24, 2025, <https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Scroll-driven_animations>
12. Scroll-driven animation timelines - CSS - MDN Web Docs, доступ отримано грудня 24, 2025, <https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Scroll-driven_animations/Timelines>
13. Core Web Vitals Thresholds: SEO Impact - Upward Engine, доступ отримано грудня 24, 2025, <https://upwardengine.com/core-web-vitals-thresholds-seo-impact/>
14. How to improve Core Web Vitals in 2025: A complete guide - OWDT, доступ отримано грудня 24, 2025, <https://owdt.com/insight/how-to-improve-core-web-vitals/>
15. The Most Important Core Web Vitals Metrics in 2025 - NitroPack, доступ отримано грудня 24, 2025, <https://nitropack.io/blog/most-important-core-web-vitals-metrics/>
16. Material Design 3 - Google's latest open source design system, доступ отримано грудня 24, 2025, <https://m3.material.io/>
17. Material 3 Expressive deep dive: All you need to know about Android's big design upgrade, доступ отримано грудня 24, 2025, <https://www.androidauthority.com/google-material-3-expressive-features-changes-availability-supported-devices-3556392/>
18. Features and APIs | Android Developers, доступ отримано грудня 24, 2025, <https://developer.android.com/about/versions/16/features>
19. Behavior changes: Apps targeting Android 16 or higher, доступ отримано грудня 24, 2025, <https://developer.android.com/about/versions/16/behavior-changes-16>
20. Window size classes - Applying layout – Material Design 3, доступ отримано грудня 24, 2025, <https://m3.material.io/foundations/layout/applying-layout/window-size-classes>
21. Compose Material 3 Adaptive | Jetpack - Android Developers, доступ отримано грудня 24, 2025, <https://developer.android.com/jetpack/androidx/releases/compose-material3-adaptive>
22. WCAG 2.2: What You Need to Know in 2026 - accessiBe, доступ отримано грудня 24, 2025, <https://accessibe.com/blog/knowledgebase/wcag-two-point-two>
23. New WCAG 3.0 in September 2025: What Changed, How It Differs from WCAG 2.x, and What To Do Next | by Nick Chukreiev | Medium, доступ отримано грудня 24, 2025, <https://medium.com/@chukreiev/new-wcag-3-0-in-september-2025-what-changed-how-it-differs-from-wcag-2-x-and-what-to-do-next-fc13824aa821>
24. Android 16 features and changes list - Android Developers, доступ отримано грудня 24, 2025, <https://developer.android.com/about/versions/16/summary>
25. How to Enable Better Text Outlining on Android 16 - YouTube, доступ отримано грудня 24, 2025, <https://www.youtube.com/watch?v=4CACpeBhyJ8>