ChatGPT

Product Requirements Document: Modular Front-End Architecture for Webflow Projects

1. Introduction

This Product Requirements Document (PRD) defines a modular, **event-driven front-end architecture** for Webflow sites using GSAP's visual animation tools. The goal is to improve **performance**, **accessibility**, **reusability**, and **developer ergonomics** while minimizing the footprint in Webflow's designer. Each module in this architecture is self-contained, load-only-when-needed, and interoperates via semantic **custom events**. A single script loader controls versioning and facilitates local development.

The design addresses the following core requirements:

- **Performance**: Code only runs when the relevant DOM exists. Scroll locking and media playback are handled efficiently, and all animations are delegated to Webflow's GSAP UI to offload heavy work from JavaScript.
- **Reusability**: Every file exports functions that can be instantiated multiple times or configured via parameters (e.g., selecting different root elements). Event names are namespaced and documented, promoting consistency.
- **Accessibility (a11y)**: All interactive components implement keyboard controls, ARIA attributes, focus management, and `inert` for background content when overlays are active.
- **Code Hygiene**: Each file has clear responsibilities, descriptive naming, safe defaults, error handling, and inline comments describing control flow. Debug logging is toggleable via a global `env.debug` flag.

This PRD assumes GSAP animations will be defined through Webflow's Interactions ("IX3") panel. The JavaScript layer simply emits and listens to **Custom Events** to trigger those animations. No GSAP code is embedded here; all timelines are configured visually by designers.

2. Global Loader

2.1 Purpose

To ensure **zero blocking** on page load, a global loader script is inserted into the Webflow project's **Footer Code** field. It loads a bundled JavaScript file (`app.min.js` for production, `app.js` for local development) **asynchronously**. The loader detects whether the site is running on localhost (or a `.local` domain) and points to the appropriate script URL.

2.2 Loader Snippet

Embed the following snippet once in **Project Settings → Footer Code**:

```
<script>
!function(d,s){
  s=d.createElement('script'); s.defer=1;
  s.src = (location.hostname==='localhost' ||
location.hostname.endsWith('.local'))
    ? 'http://localhost:5173/app.js'  // local dev bundle
    : 'https://cdn.yoursite.com/app.min.js'; // production bundle
  d.head.appendChild(s);
}(document);
</script>
```

The script sets `defer` to avoid blocking parsing. When executed, the bundle prints a version message to the console: `[APP] loaded vX.Y.Z`. A semantic version string is exposed via `window.APP.version` for debugging.

3. Directory Layout

The project uses an **ES module** structure under `src/`. Build tools (e.g., Vite or Rollup) bundle these files into the final `app.min.js`. Every module exports functions for initialization; nothing runs at import time except constant declarations.

```
src/
  core/
    env.js              # environment flags (debug, design mode, prefers-
reduced-motion) and safe logger
    events.js           # generic emit() helper, bridging to Webflow IX3
    scrolllock.js       # lock/unlock page scroll, manages stack and inert
state
  modules/
    accordion.js        # two-level accordion component
    lightbox.js         # single lightbox component that sources content from
slide data attributes
    vimeo.js            # lightweight Vimeo player embed helper
    parallax.js         # optional: set CSS variable per slide for parallax
effects
    snapsnap.js         # helpers for enabling/disabling scroll-snap
temporarily
    preloader.js        # preloader routine (locks scroll until timeline
completes)
    menu.js             # navigation drawer and page transition helper
  index.js              # orchestrator: initializes modules conditionally and
logs boot

# build/
#   app.min.js          # compiled, minified bundle served in production
```

3.1 Reusable API Pattern

- Each module exports only what is needed (typically a single `initXXX` function). Variables internal to the module are `const` or `let` scoped.

- Modules accept **configuration objects** so selectors and behavior can be customized (`initLightbox({root:'#project-lightbox', closeDelayMs:1000})`). Sensible defaults are provided.
- All event names are centralized (declared near the top of each module) to avoid typos and ease documentation.
- Error and absence conditions are logged via `env.log()` or `console.warn()` when running in debug mode.

4. Core Modules

4.1 `env.js`

Defines global environment flags and a safe logger. It determines whether the page is open in Webflow designer (`.w-editor` class), whether the user prefers reduced motion, and toggles debug output:

```
// src/core/env.js
export const env = {
  debug: true,  // set false in production build
  isDesigner: document.documentElement.classList.contains('w-editor'),
  reduced: window.matchMedia('(prefers-reduced-motion: reduce)').matches
};

/**
 * Safe logger – prints only when env.debug is true.
 * Usage: log('[MODULE]', ...args)
 */
export const log = (...args) => {
  if (env.debug) console.log(...args);
};
```

`env.reduced` can be used to bypass animation code and immediately reveal states. Designers can toggle `env.debug` to silence logs in production.

4.2 `events.js`

Provides a unified method to dispatch both Webflow IX3 custom events and native `CustomEvent` instances. This ensures GSAP UI triggers listen correctly regardless of execution context.

```
// src/core/events.js
import { env } from './env.js';

/**
 * Emit a custom event.  This helper:
 *   1. Calls Webflow IX3 emit if available (animations in the UI can listen
on these names).
 *   2. Dispatches a bubbling CustomEvent on a target element (default:
document).
 *   3. Logs the event when env.debug is true.
 *
 * @param {string} name   The event name (used by both IX3 and CustomEvent)
 * @param {EventTarget} [target=document] The DOM element to dispatch the
event on
 * @param {Object} [detail={}] Additional data passed to the event detail
 */
export function emit(name, target = document, detail = {}) {
  try {
    const ix = window.Webflow?.require?.('ix3');
    ix?.emit?.(name);
  } catch (e) {
    // swallow silently if IX3 is unavailable (common on plain pages)
  }
  target.dispatchEvent(new CustomEvent(name, { bubbles: true, detail }));
  if (env.debug) console.log('[EVENT]', name, target, detail);
}
```

4.3 `scrolllock.js`

Implements a **stack-based scroll lock** that supports nested locks (e.g. preloader overlay inside a lightbox). When any component locks the scroll, the body is fixed and the current scroll position is preserved. The module also manages the `inert` attribute on siblings of the overlay to prevent interaction outside modals.

```
// src/core/scrolllock.js
let locks = 0;
let savedY = 0;
let prevScrollBehavior = '';

/**
 * Apply inert to all siblings of the overlay.  Called when the first lock is
applied.
 */
function setSiblingsInert(on, overlay) {
  const siblings = Array.from(document.body.children).filter(el => el !==
overlay);
  siblings.forEach(el => on ? el.setAttribute('inert', '') :
el.removeAttribute('inert'));
}

/**
 * Lock page scroll and apply inert on siblings.
 * @param {HTMLElement} [overlay] The overlay element to exclude from inert;
optional
 */
export function lockScroll(overlay) {
  if (locks++ > 0) return;
  const html = document.documentElement;
  prevScrollBehavior = html.style.scrollBehavior;
  html.style.scrollBehavior = 'auto';
  savedY = window.scrollY || html.scrollTop;
  Object.assign(document.body.style, {
    position: 'fixed',
    top: `-${savedY}px`,
    left: '0',
    right: '0',
    width: '100%',
    overflow: 'hidden',
    overscrollBehavior: 'none'
  });
  if (overlay) setSiblingsInert(true, overlay);
  console.log('[SCROLL] locked');
}

/**
 * Unlock page scroll.  Restores original scroll position and inert state.
Uses optional delay.
 * @param {Object} [opts]
 * @param {number} [opts.delayMs=0] Duration (ms) to wait before unlocking.
 * @param {HTMLElement} [overlay] The overlay element whose siblings had
inert applied
 */
export function unlockScroll(opts = {}, overlay) {
  const { delayMs = 0 } = opts;
  const run = () => {
    if (--locks > 0) return;
    const html = document.documentElement;
    Object.assign(document.body.style, {
      position: '',
      top: '',
      left: '',
      right: '',
      width: '',
      overflow: '',
      overscrollBehavior: ''
    });
    html.style.scrollBehavior = prevScrollBehavior || '';
    window.scrollTo(0, savedY);
    if (overlay) setSiblingsInert(false, overlay);
    console.log('[SCROLL] unlocked');
  };
  delayMs ? setTimeout(run, delayMs) : run();
}
```

Usage: Components (e.g. lightbox, preloader) call `lockScroll(overlay)` when opening and `unlockScroll({delayMs}, overlay)` when closing. If multiple overlays are present (nested), the lock stack ensures that scroll is only restored when all locks have been released.

5. Feature Modules

5.1 Accordion

The accordion supports **two nested levels** (level-1 and level-2). Each level can be toggled independently, but opening a level-1 panel automatically closes all open level-2 panels. The accordion emits **four custom events** for GSAP panels:

- `ACC_L1_OPEN` / `ACC_L1_CLOSE` on `.accordeon-item--level1`
- `ACC_L2_OPEN` / `ACC_L2_CLOSE` on `.accordeon-item--level2`

5.1.1 Markup Requirements

- Container: `.accordeon` (root element)
- Items: `.accordeon-item--level1` and `.accordeon-item--level2`
- Trigger: Each item must contain a native `<button>` element with the class `.accordeon__trigger` **inside a heading** (e.g. `<h2>` for level-1 or `<h3>` for level-2 items). Using a real button ensures proper semantics and focusability. Place any decorative icons in CSS or mark them with `aria-hidden="true"` so they are not announced by screen readers.
- Panel: `.accordeon__list` (immediately inside each item, collapsed by default). When the panel is visible, give it `role="region"` and `aria-labelledby` pointing to the trigger's ID. This identifies the expanded content as a landmark region to assistive technologies.

5.1.2 Accessibility

- Each trigger has `role="button"`, `tabindex="0"`, `aria-controls` (pointing to the associated panel ID), and `aria-expanded` toggled by JS.
- Keyboard support: `Enter` or `Space` triggers a toggle. `Tab` navigation cycles through focusable elements naturally.
- Panels use `max-height` transitions; open panels set `max-height: none` after transition to avoid reflow on content changes. A `ResizeObserver` updates open panels when images load.

5.1.3 API and Behavior

```
// src/modules/accordion.js
import { emit } from '../core/events.js';

export function initAccordion(rootSelector = '.accordeon') {
  const root = document.querySelector(rootSelector);
  if (!root) { console.log('[ACCORDION] root not found'); return; }

  // Utility functions for level detection and retrieving panels
  const isL1 = el => el.classList.contains('accordeon-item--level1');
  const isL2 = el => el.classList.contains('accordeon-item--level2');
  const panelOf = item => item.querySelector(':scope > .accordeon__list');
  const groupOf = item => isL1(item) ? root :
item.closest('.accordeon__list');

  // Initialize ARIA attributes on triggers
  root.querySelectorAll('.accordeon__trigger').forEach((trigger, idx) => {
    trigger.setAttribute('role', 'button');
    trigger.setAttribute('tabindex', '0');
    const item = trigger.closest('.accordeon-item--level1, .accordeon-item--
level2');
    const panel = panelOf(item);
    if (panel) {
      const pid = panel.id || `accordion-panel-${idx}`;
      panel.id = pid;
      trigger.setAttribute('aria-controls', pid);
      trigger.setAttribute('aria-expanded', 'false');
    }
  });

  // Transition helpers: open/collapse panels using max-height
  function expand(panel) {
    panel.style.maxHeight = panel.scrollHeight + 'px';
    panel.dataset.state = 'opening';
    const onEnd = (e) => {
      if (e.propertyName !== 'max-height') return;
      panel.removeEventListener('transitionend', onEnd);
      if (panel.dataset.state === 'opening') {
        panel.style.maxHeight = 'none';
        panel.dataset.state = 'open';
      }
    };
    panel.addEventListener('transitionend', onEnd);
  }
  function collapse(panel) {
    const currentHeight = panel.style.maxHeight === 'none' ?
panel.scrollHeight : parseFloat(panel.style.maxHeight || 0);
    panel.style.maxHeight = (currentHeight || panel.scrollHeight) + 'px';
    panel.offsetHeight; // force reflow
    panel.style.maxHeight = '0px';
    panel.dataset.state = 'closing';
    const onEnd = (e) => {
      if (e.propertyName !== 'max-height') return;
      panel.removeEventListener('transitionend', onEnd);
      panel.dataset.state = 'collapsed';
    };
    panel.addEventListener('transitionend', onEnd);
  }

  // Close sibling panels within the same group (level-1 or level-2)
  function closeSiblings(item) {
    const group = groupOf(item);
    if (!group) return;
    const targetClass = isL1(item) ? 'accordeon-item--level1' : 'accordeon-
item--level2';
    Array.from(group.children).forEach(sibling => {
      if (sibling === item || !sibling.classList.contains(targetClass))
return;
```

```
      const panel = panelOf(sibling);
      if (panel && (panel.dataset.state === 'open' || panel.dataset.state === 'opening')) {
        collapse(panel);
        sibling.querySelector('.accordeon__trigger')?.setAttribute('aria-expanded', 'false');
        emit(isL1(item) ? 'ACC_L1_CLOSE' : 'ACC_L2_CLOSE', sibling, { source: 'sibling' });
      }
    });
  }

  // Reset all level-2 panels (called when a level-1 panel is toggled)
  function resetAllL2() {
    root.querySelectorAll('.accordeon-item--level2 .accordeon__list').forEach(panel => {
      if (panel.dataset.state === 'open' || panel.dataset.state === 'opening') {
        collapse(panel);
        const item = panel.closest('.accordeon-item--level2');
        item?.querySelector('.accordeon__trigger')?.setAttribute('aria-expanded', 'false');
        emit('ACC_L2_CLOSE', item, { source: 'reset-all' });
      }
    });
  }

  // Toggle panel: open if collapsed, close if open
  function toggle(item) {
    const panel = panelOf(item);
    if (!panel) return;
    const trigger = item.querySelector('.accordeon__trigger');
    const opening = !(panel.dataset.state === 'open' || panel.dataset.state === 'opening');
    closeSiblings(item);
    if (opening && isL1(item)) resetAllL2();
    if (opening) {
      expand(panel);
      trigger?.setAttribute('aria-expanded', 'true');
      emit(isL1(item) ? 'ACC_L1_OPEN' : 'ACC_L2_OPEN', item, { opening: true });
    } else {
      collapse(panel);
      trigger?.setAttribute('aria-expanded', 'false');
      if (isL1(item)) resetAllL2();
      emit(isL1(item) ? 'ACC_L1_CLOSE' : 'ACC_L2_CLOSE', item, { opening: false });
    }
  }

  // Initial state: collapse all panels, mark state
  document.body.classList.add('js-prep');
  root.querySelectorAll('.accordeon__list').forEach(panel => {
    panel.style.maxHeight = '0px';
    panel.dataset.state = 'collapsed';
  });
  requestAnimationFrame(() => document.body.classList.remove('js-prep'));
  console.log('[ACCORDION] init');

  // Event handlers
  root.addEventListener('click', e => {
    const trigger = e.target.closest('.accordeon__trigger');
    if (!trigger || !root.contains(trigger)) return;
    e.preventDefault();
    const item = trigger.closest('.accordeon-item--level1, .accordeon-item--
level2');
    if (item) toggle(item);
  });
  root.addEventListener('keydown', e => {
    const trigger = e.target.closest('.accordeon__trigger');
    if (!trigger || !root.contains(trigger)) return;
    if (e.key === 'Enter' || e.key === ' ') {
      e.preventDefault();
      const item = trigger.closest('.accordeon-item--level1, .accordeon-
item--level2');
      if (item) toggle(item);
    }
  });

  // ResizeObserver: adjust height when images or dynamic content load into
an open panel
  const ro = new ResizeObserver(entries => {
    entries.forEach(({ target: panel }) => {
      if (panel.dataset.state === 'open') {
        panel.style.maxHeight = 'none';
      } else if (panel.dataset.state === 'opening') {
        panel.style.maxHeight = panel.scrollHeight + 'px';
      }
    });
  });
  root.querySelectorAll('.accordeon__list').forEach(panel =>
ro.observe(panel));
}
```

5.1.4 CSS Requirements

Place the following rules in your global CSS (loaded once). The `.js-prep` class disables transitions during initialization to prevent janky collapse on page load.

```
.accordeon__list {
  overflow: hidden;
  transition: max-height 0.28s cubic-bezier(0.25, 0.8, 0.25, 1);
  will-change: max-height;
}
.js-prep .accordeon__list { transition: none !important; }
.accordeon__trigger { cursor: pointer; user-select: none; }
```

In addition, global CSS (see section 7) defines the focus outline and overscroll behavior.

5.2 Lightbox

The lightbox displays external video content (typically Vimeo) or other data from slides. Only one instance is supported on a page. The lightbox is designed to be **accessible**, **keyboard-friendly**, and safe to re-open repeatedly.

5.2.1 Markup Requirements

- Root: `#project-lightbox` – a container (e.g. `<div>`) with `role="dialog"`, `aria-modal="true"`, and `aria-hidden="true"`. To give the dialog an **accessible name**, add either an `aria-label` directly on the container (e.g. `aria-label="Project details"`) or reference a heading inside the dialog via `aria-labelledby`.
- Inner wrapper: `.project-lightbox__inner` – the actual dialog content area (used for outside-click detection and focus management). Place a heading inside this wrapper (e.g. `<h2 id="lightboxTitle">Project title</h2>`) so screen readers announce the purpose of the dialog. Reference this heading from the root with `aria-labelledby="lightboxTitle"`. If the dialog contains descriptive text, reference it via `aria-describedby`.
- Video area: `.video-area` – container into which a `<iframe>` is injected by `mountVimeo()`. For accessibility, set a title attribute on the `<iframe>` describing the video's content (e.g. `<iframe title="Project demo video" ...>`). This helps screen reader users understand the purpose of the embedded media.
- Slides: any elements with class `.slide` inside `.scroll-wrapper`. Each slide must define data attributes:
- `data-video`: the Vimeo URL or ID (numeric)
- `data-title`: optional title
- `data-text`: optional description

Additionally, each slide should include visible heading text inside its `.text-box` to provide a semantic heading structure for assistive technologies. The heading level should reflect the document outline (e.g. `<h2>` for top-level slides).

5.2.2 API and Behavior

```
// src/modules/lightbox.js
import { emit } from '../core/events.js';
import { lockScroll, unlockScroll } from '../core/scrolllock.js';
import { mountVimeo } from './vimeo.js';

export function initLightbox({ root = '#project-lightbox', closeDelayMs =
1000 } = {}) {
  const lightbox = document.querySelector(root);
  if (!lightbox) { console.log('[LIGHTBOX] not found'); return; }
  const videoArea = lightbox.querySelector('.video-area');
  const slides = document.querySelectorAll('.scroll-wrapper .slide');
  const inner = lightbox.querySelector('.project-lightbox__inner');
  let isOpen = false;
  let lastFocus = null;

  // Helper: apply inert on page siblings (only first lock uses inert)
  function setInert(on) {
    Array.from(document.body.children).filter(n => n !== lightbox)
      .forEach(n => on ? n.setAttribute('inert', '') :
n.removeAttribute('inert'));
  }

  // Focus trap: cycle focus within the lightbox when pressing Tab
  function trapFocus(e) {
    if (e.key !== 'Tab') return;
    const focusables = lightbox.querySelectorAll(
      'a[href], button, input, select, textarea,
[tabindex]:not([tabindex="-1"])'
    );
    const list = Array.from(focusables).filter(el => !
el.hasAttribute('disabled') && !el.getAttribute('aria-hidden'));
    if (list.length === 0) { e.preventDefault(); lightbox.focus(); return; }
    const first = list[0], last = list[list.length - 1];
    if (e.shiftKey && document.activeElement === first) {
e.preventDefault(); last.focus(); }
    else if (!e.shiftKey && document.activeElement === last) {
e.preventDefault(); first.focus(); }
  }

  // Open the lightbox using data from a slide
  function openFromSlide(slide) {
    if (isOpen) return;
    isOpen = true;
    lastFocus = document.activeElement instanceof HTMLElement ?
document.activeElement : null;
    // Extract data attributes
    const video = slide.dataset.video || '';
    const title = slide.dataset.title || '';
    const text = slide.dataset.text || '';
    // Mount Vimeo player
    if (videoArea) {
      const id = String(video).match(/\d+/)?.[0] || video;
      // Pass privacy flags; add dnt=1 to minimize tracking
      mountVimeo(videoArea, id, { autoplay: 1, muted: 1, controls: 0,
background: 1, playsinline: 1, dnt: 1 });
    }
    lightbox.setAttribute('aria-hidden', 'false');
    lightbox.setAttribute('data-open', 'true');
    setInert(true);
    lockScroll(lightbox);
    // Focus the inner wrapper for accessibility
    lightbox.setAttribute('tabindex', '-1');
    (inner || lightbox).focus();
    // Emit event so GSAP UI can play the open animation
    emit('LIGHTBOX_OPEN', lightbox, { video, title, text });
  }

  // Request closing the lightbox
  function requestClose() {
    if (!isOpen) return;
    // Emit close event to drive GSAP UI
    emit('LIGHTBOX_CLOSE', lightbox);
    // Unlock scroll after animation (fallback uses closeDelayMs)
    unlockScroll({ delayMs: closeDelayMs }, lightbox);
    lightbox.setAttribute('aria-hidden', 'true');
    lightbox.removeAttribute('data-open');
    setInert(false);
    // Remove video to stop playback
    if (videoArea) videoArea.innerHTML = '';
    // Restore focus to previously focused element
    if (lastFocus && document.body.contains(lastFocus)) {
      lastFocus.focus();
    }
    isOpen = false;
  }

  // Bind click handlers on slides
  slides.forEach(slide => {
    slide.addEventListener('click', () => openFromSlide(slide));
  });
  // Close when clicking outside the inner content
  lightbox.addEventListener('click', e => {
    // If the click is outside .project-lightbox__inner, close
    if (inner && !e.target.closest('.project-lightbox__inner'))
requestClose();
    else if (!inner && e.target === lightbox) requestClose();
  });
  // Keyboard handlers
  document.addEventListener('keydown', e => {
    if (!isOpen) return;
    if (e.key === 'Escape') requestClose();
    else if (e.key === 'Tab') trapFocus(e);
  });
  // Optional: unlock scroll when custom event dispatched at end of GSAP
close timeline
  lightbox.addEventListener('LIGHTBOX_CLOSED_DONE', () => unlockScroll({},
lightbox));
  console.log('[LIGHTBOX] init');
}
```

5.2.3 Additional Notes

- The component uses `inert` (supported widely) to prevent interaction with the background when the lightbox is open.
- Passing `dnt=1` to Vimeo's player instructs it not to store cookies.
- If the user prefers reduced motion (`env.reduced`), skip dispatching open/close events and simply show/hide the lightbox instantly, unlocking scroll immediately.

5.3 Vimeo Helper

The `vimeo.js` module wraps the Vimeo player embed logic. It accepts either a numeric ID or a full URL and constructs the appropriate embed. Default options enable autoplay, mute, controls hidden, background mode, and playsinline behavior.

```
// src/modules/vimeo.js
export function mountVimeo(el, idOrUrl, opts = { autoplay:1, muted:1,
controls:0, background:1, playsinline:1 }) {
  const id = String(idOrUrl).match(/\d+/)?.[0] || idOrUrl;
  const qs = new URLSearchParams({ ...opts, dnt: 1 });
  el.innerHTML = `\n<iframe src="https://player.vimeo.com/video/${id}?$
{qs}"\n  allow="autoplay; fullscreen; picture-in-picture" allowfullscreen></
iframe>\n`;
}
```

This helper intentionally does not rely on any external libraries (e.g. Vimeo Player API) to keep the bundle size minimal. For advanced control (e.g. events, seeking) you may later replace this with Vimeo's SDK.

5.4 Preloader

Some pages require a **preloader sequence** to ensure assets are primed before the user interacts. The preloader listens for a `[data-preloader]` element and emits two events:

- `PRELOADER_START` – triggers the GSAP panel to play the intro animation.
- `PRELOADER_DONE` – triggers the outro; when done, scroll is unlocked.

```
// src/modules/preloader.js
import { emit } from '../core/events.js';
import { lockScroll, unlockScroll } from '../core/scrolllock.js';

export function runPreloader({ minMs = 800 } = {}) {
  const el = document.querySelector('[data-preloader]');
  if (!el) { return; }
  console.log('[PRELOADER] init');
  lockScroll(el);
  emit('PRELOADER_START', el);
  const start = performance.now();
  // Unlock scroll when the GSAP panel fires PRELOADER_DONE
  el.addEventListener('PRELOADER_DONE', () => unlockScroll({}, el), { once:
true });
```

```
    // Fallback: auto emit PRELOADER_DONE if panel does not dispatch
    setTimeout(() => {
      if (!el.hasAttribute('data-done')) {
        emit('PRELOADER_DONE', el);
        unlockScroll({}, el);
      }
    }, Math.max(0, minMs - (performance.now() - start)));
  }
```

The GSAP panel should set `data-done` on the preloader element (via UI) or dispatch `PRELOADER_DONE` at the end of the animation. If not, the fallback ensures the preloader finishes after `minMs`.

Accessibility Considerations

- **Use `aria-busy`**: While the preloader is active, set `aria-busy="true"` on the `<main>` or relevant container to signal that content is loading. When the preloader completes, remove the attribute.
- **Provide descriptive feedback**: Include off-screen text within the preloader (e.g. `<span class="visuallyhidden">Loading…</span>`) so screen readers announce that content is loading. Ensure this text is removed or hidden after the preloader finishes.
- **Allow skipping**: Long or decorative preloaders can frustrate users; provide a “Skip intro” link or button that immediately triggers `PRELOADER_DONE`. This link should be focusable and clearly labelled (e.g. `<button data-preloader-skip aria-label="Skip loading animation">Skip</button>`).

5.5 Menu (Navigation Drawer)

The menu module handles opening/closing a navigation drawer and facilitates page transitions. It emits `MENU_OPEN` and `MENU_CLOSE` events for the drawer’s GSAP animations. When a link inside the drawer is clicked, the JS waits for the close animation to finish (listening for `MENU_CLOSED_DONE`) before navigating to the new page. A fallback timeout ensures navigation occurs even if the event isn’t fired.

Markup and Accessibility Guidelines

- Wrap your navigation drawer in a semantic `<nav>` element and apply the class `.nav-drawer` (e.g. `<nav class="nav-drawer" aria-label="Main menu">...</nav>`). The `aria-label` provides an accessible name for screen reader users. Alternatively, you can reference a visible heading using `aria-labelledby`.
- Structure the menu items with an unordered list (`<ul><li><a href="...">…</a></li>…</ul>`). This conveys the count and sequence of items to assistive technologies.
- Indicate the current page by adding `aria-current="page"` to the link representing the active route. This helps users orient themselves within the site.
- Ensure that keyboard users can open and close the drawer via button controls that are real `<button>` elements. The buttons should have `data-menu-open` or `data-menu-close` attributes and descriptive accessible names (e.g. `<button data-menu-open aria-label="Open menu">≡</button>`).

```
// src/modules/menu.js
import { emit } from '../core/events.js';

export function initMenu({ drawerSel = '.nav-drawer' } = {}) {
  const drawer = document.querySelector(drawerSel);
  if (!drawer) { return; }
  console.log('[MENU] init');
  // Buttons to open/close the drawer
  document.querySelectorAll('[data-menu-open]').forEach(btn =>
btn.addEventListener('click', () => emit('MENU_OPEN', drawer)));
  document.querySelectorAll('[data-menu-close]').forEach(btn =>
btn.addEventListener('click', () => emit('MENU_CLOSE', drawer)));
  // Intercept link clicks inside the drawer
  drawer.addEventListener('click', e => {
    const a = e.target.closest('a[href]');
    if (!a) return;
    e.preventDefault();
    const href = a.getAttribute('href');
    emit('MENU_CLOSE', drawer);
    const navigate = () => { location.href = href; };
    // If GSAP panel fires MENU_CLOSED_DONE, navigate at that moment
    drawer.addEventListener('MENU_CLOSED_DONE', navigate, { once: true });
    // Fallback navigation after ~700ms (sync with animation duration)
    setTimeout(navigate, 700);
  });
}
```

5.6 Snap Helpers (Optional)

These helpers temporarily disable scroll-snap during programmatic transitions (e.g. when a lightbox opens). Snap settings should be restored once the transition ends.

```
// src/modules/snapsnap.js
export function setSnap(el, mode) { el.style.scrollSnapType = mode; }
export function tempDisableSnap(el, ms = 400) {
  const prev = getComputedStyle(el).scrollSnapType;
  el.style.scrollSnapType = 'none';
  setTimeout(() => el.style.scrollSnapType = prev, ms);
}
```

5.7 Parallax (Optional)

If you prefer code-driven parallax instead of Webflow's GSAP "While Scrolling" triggers, implement an IntersectionObserver that sets a CSS custom property (e.g. `--p`) on each slide based on scroll progress. GSAP UI can then map this variable to transforms. This module is not required if you use Webflow's built-in scroll triggers.

6. Orchestrator (`index.js`)

The orchestrator runs on `DOMContentLoaded` and initializes modules only when the corresponding DOM elements exist. It also surfaces the application version and debug flag via `window.APP`.

```
// src/index.js
import { env } from './core/env.js';
import { initAccordion } from './modules/accordion.js';
import { initLightbox } from './modules/lightbox.js';
import { runPreloader } from './modules/preloader.js';
import { initMenu } from './modules/menu.js';

// Expose version and debug globally
window.APP = { version: '0.1.0', debug: env.debug };

document.addEventListener('DOMContentLoaded', () => {
  console.log('[APP] loaded v' + APP.version);
  if (document.querySelector('.accordeon')) {
    console.log('[BOOT] accordion');
    initAccordion('.accordeon');
  }
  if (document.querySelector('#project-lightbox')) {
    console.log('[BOOT] lightbox');
    initLightbox({ root: '#project-lightbox', closeDelayMs: 1000 });
  }
  if (document.querySelector('[data-preloader]')) {
    console.log('[BOOT] preloader');
    runPreloader({ minMs: 800 });
  }
  if (document.querySelector('.nav-drawer')) {
    console.log('[BOOT] menu');
    initMenu({ drawerSel: '.nav-drawer' });
  }
});
```

This approach ensures modules are not executed on pages that do not need them, improving performance. Future modules can be registered in similar conditional blocks.

7. Global CSS and Polyfills

Place these rules in a global stylesheet to improve baseline experience:

```
html, body { overscroll-behavior: none; }
* { -webkit-tap-highlight-color: transparent; }
:focus-visible { outline: 2px solid currentColor; outline-offset: 2px; }
[hidden] { display: none !important; }
/* Scroll snap container and slide defaults */
.scroll-wrapper {
  height: 100dvh;
  overflow-y: auto;
  overflow-x: hidden;
  scroll-snap-type: y proximity;
  overscroll-behavior: contain;
  touch-action: pan-y;
  scrollbar-gutter: stable both-edges;
}
.slide {
  height: 100dvh;
  width: 100vw;
  scroll-snap-align: center;
  scroll-snap-stop: always;
  contain: layout paint;
}
.slide .text-box {
  height: 100svh;
  padding-bottom: calc(6vh + env(safe-area-inset-bottom));
}
.slide .bg-wrap {
  position: absolute;
  inset: auto 0 0 0;
  top: -10dvh;
  height: calc(100dvh + 20dvh);
}
```

7.1 Polyfills

- Add `<link rel="preconnect" href="https://player.vimeo.com">` and `<link rel="preconnect" href="https://i.vimeocdn.com">` in the page `<head>` to speed up video loads.
- To support older browsers that lack `inert`, include the WICG inert polyfill (optional). Place it before your bundle.

8. Performance and Reusability Considerations

- **Lazy Initialization**: Modules only run if the matching selector exists. This avoids unnecessary DOM queries and event listeners.
- **Debug vs Production**: Build tools should replace `env.debug` with `false` and remove console logs in production. Consider using environment variables to set this at build time.
- **Modular Design**: Each feature is encapsulated; adding new modules does not impact existing ones. Use clear naming conventions for events and keep event payloads consistent.
- **Reduced Motion**: When `env.reduced` is true, bypass GSAP triggers and show/hide states instantly. Modules should check this flag and react accordingly.
- **Error Handling**: All modules check DOM presence and log warnings rather than throwing exceptions. This ensures that misconfigured pages do not break the entire bundle.
- **Accessibility**: Use ARIA roles and attributes consistently. Provide keyboard interactions for all interactive elements. Use `inert` to disable background interactions when overlays are active.

9. Documentation and Comments

- Each module file should start with a **module header comment** summarizing its purpose and exported functions.
- Within functions, comment complex logic or non-obvious decisions. Avoid obvious “this toggles X” comments.
- Document all **Custom Events** in a dedicated section of your repository (e.g. `docs/events.md`) so designers know which names to listen to in GSAP UI.
- Use descriptive variable names (`isOpen`, `trigger`, `panelOf`) over abbreviations. This reduces cognitive overhead.
- Include JSDoc annotations for exported functions, especially if third parties will consume them.

10. Appendix: Custom Event Reference

Event Name	Emitted On	Description
`ACC_L1_OPEN`	`.accordeon-item--level1`	Fired when a level-1 accordion panel opens. GSAP UI attaches open animation timeline here.
`ACC_L1_CLOSE`	`.accordeon-item--level1`	Fired when a level-1 panel closes. Used to reverse animations and reset nested panels.
`ACC_L2_OPEN`	`.accordeon-item--level2`	Fired when a level-2 accordion panel opens.
`ACC_L2_CLOSE`	`.accordeon-item--level2`	Fired when a level-2 panel closes.
`LIGHTBOX_OPEN`	`#project-lightbox`	Fired just after the lightbox becomes visible. Contains `detail: {video, title, text}`. GSAP UI should animate fade/scale in.
`LIGHTBOX_CLOSE`	`#project-lightbox`	Fired when closing is requested. GSAP UI should animate fade out. When animation ends, dispatch `LIGHTBOX_CLOSED_DONE`.
`LIGHTBOX_CLOSED_DONE`	`#project-lightbox`	Dispatched at the end of the close animation. The JS listens for this to unlock scroll (alternative to timeout).
`PRELOADER_START`	`[data-preloader]`	Fired at page load to play intro preloader animation.
`PRELOADER_DONE`	`[data-preloader]`	Fired when preloader finishes. JS unlocks scroll here.
`MENU_OPEN`	`.nav-drawer`	Fired when the navigation drawer is opened via UI control.
`MENU_CLOSE`	`.nav-drawer`	Fired when closing the drawer (e.g. clicking outside or a link). JS waits for `MENU_CLOSED_DONE` before navigation.

Event Name	Emitted On	Description
`MENU_CLOSED_DONE`	`.nav-drawer`	Dispatched at the end of the drawer close timeline. JS uses this to navigate to a new URL.

Each event supports additional properties in `detail` if needed. Designers and developers should maintain this table to avoid naming conflicts.

This PRD provides a clear roadmap for implementing a **performant**, **accessible**, and **maintainable** front-end architecture for Webflow projects. It emphasizes separation of concerns, event-driven communication, and robust error handling. Developers can follow the outlined modules and guidelines to scaffold the codebase efficiently, while designers can rely on documented events to orchestrate animations in GSAP UI without touching JavaScript.

11. Potential Optimisations and Simplifications

The code examples in this PRD are deliberately explicit to aid comprehension. Once you have the initial system working, several opportunities exist to **streamline the implementation** without altering behaviour or breaking the event contract:

- **Centralise `inert` handling**: Currently both `scrolllock.js` and the lightbox manage the `inert` attribute on page siblings. Consider moving `setSiblingsInert()` into `scrolllock.js` and accepting an optional overlay argument. Then you can remove the custom `setInert()` helper from the lightbox, reducing duplication.
- **Simplify state flags**: The lightbox uses both `isOpen` and `openGuard` (in the original plan) to prevent double opens. A single boolean (`isOpen`) is sufficient to track whether the overlay is active.
- **Inline small helpers**: If you do not anticipate extending the Vimeo API, you can fold `mountVimeo()` directly into the lightbox module. Eliminating the extra import reduces overhead and file count.
- **Consolidate accordion loops**: Functions like `resetAllL2()` and `closeSiblings()` both traverse nested panels. Refactoring them into a single loop improves readability and avoids repeated DOM queries.
- **Use class toggles for scroll locking**: Instead of assigning a block of inline styles to `body` on each lock, you could toggle a `.scroll-locked` class and define the fixed positioning in CSS. This shortens the JS and makes it easier to adjust the styling later.
- **Unify logging**: Route all console output through the `log()` helper defined in `env.js` so that setting `env.debug` to `false` truly silences all messages across modules.

These refinements are optional; the provided code works as written. They are offered here to guide future iterations or encourage even cleaner abstractions.

12. Manual Implementation Checklist (Cursor & Webflow)

Follow this checklist to correctly integrate the modular architecture into your environment and Webflow project. Each step corresponds to either a local code change (Cursor) or a Webflow configuration.

1. **Loader Setup (Webflow)**

2. In Webflow → **Project Settings → Footer Code**, add the loader snippet from section 2.2. Update the CDN URL (`https://cdn.yoursite.com/app.min.js`) to the actual location of your compiled bundle. This ensures the script loads on every page.
3. **Directory Structure (Cursor)**
4. Create the `src/` directory with `core/` and `modules/` subdirectories. Inside, add each file listed in section 3 (`env.js`, `events.js`, `scrolllock.js`, `accordion.js`, `lightbox.js`, etc.). Copy or adapt the code examples accordingly.
5. Configure your build tool (e.g. Vite or Rollup) to bundle these modules into `app.min.js` for production and `app.js` for development. The loader snippet references these files.
6. **Version and Debug Flags (Cursor)**
7. In `index.js`, set `window.APP.version` to your real semantic version and ensure `env.debug` is `false` in production builds. This controls console logging and version reporting.
8. **Global Styles (Webflow)**
9. Insert the CSS rules from section 7 into your site's global stylesheet. This includes overscroll behaviour, tap highlight colour, focus styles, scroll-snap defaults, and safe area padding.
10. If you adopt the class-based scroll lock suggestion, define a `.scroll-locked` rule (e.g. `body.scroll-locked { position: fixed; top: 0; left: 0; right: 0; overflow: hidden; width: 100%; }`). Adjust the JS accordingly.
11. **Markup Modifications (Webflow Designer)**
12. **Accordion**: Wrap your accordion content in an element with class `.accordeon`. For each first-level item, add `.accordeon-item--level1`; for nested items, use `.accordeon-item--level2`. Within each item, designate the clickable header with `.accordeon__trigger` and the collapsible content with `.accordeon__list`.
13. **Lightbox**: Add an element with `id="project-lightbox"`, `role="dialog"`, `aria-modal="true"`, and `aria-hidden="true"`. Inside it, create `.project-lightbox__inner` and a `.video-area` container. Each `.slide` in your scroll wrapper must have `data-video`, `data-title`, and `data-text` attributes.
14. **Preloader**: Include an element with the attribute `[data-preloader]` for the preloader overlay. Style it via Webflow and wire GSAP animations to `PRELOADER_START` and `PRELOADER_DONE` events.
15. **Menu**: Create a `.nav-drawer` for your off-canvas menu. Add buttons with `[data-menu-open]` and `[data-menu-close]` attributes to open and close the drawer.
16. **GSAP UI Wiring (Webflow)**
17. In Webflow's Interactions panel, create custom event listeners using the names in the event table (section 10). Attach your GSAP timelines to `ACC_L1_OPEN`, `ACC_L1_CLOSE`, `ACC_L2_OPEN`, `ACC_L2_CLOSE`, `LIGHTBOX_OPEN`, `LIGHTBOX_CLOSE`, `PRELOADER_START`, `PRELOADER_DONE`, `MENU_OPEN`, `MENU_CLOSE`, and so on. For close animations, dispatch the corresponding `*_CLOSED_DONE` events at the last keyframe to signal the JS to continue (e.g. unlock scroll or navigate).
18. **Head Tags (Webflow)**
19. Add `<link rel="preconnect" href="https://player.vimeo.com">` and `<link rel="preconnect" href="https://i.vimeocdn.com">` to your page `<head>` to reduce latency on video loads.
20. If you need to support browsers without native `inert` support, include the [WICG inert polyfill]() before your bundle.
21. **Building and Testing (Cursor & Webflow)**
22. Build your bundle and verify that `app.min.js` and `app.js` are generated correctly. Upload `app.min.js` to your CDN and confirm that the loader fetches it in production.

23. In Webflow preview or on a staging domain, test each interaction: accordion toggling (keyboard and mouse), lightbox open/close behaviour, preloader completion, and menu navigation. Watch the browser console for errors and adjust event names or selectors if something doesn't trigger.

Performing these tasks will ensure the architecture runs smoothly in both development (Cursor) and the live Webflow site, and that all animations are triggered by the appropriate custom events.

13. Accessibility Checklist

To achieve a fully usable experience for people with disabilities, verify your implementation against these guidelines, informed by the W3C Web Content Accessibility Guidelines (WCAG) 2.1/2.2 and ARIA Authoring Practices:

- **Semantic structure**: Use native HTML elements wherever possible. Wrap accordion triggers in heading tags (`<h2>`/`<h3>`) containing a `<button>` and wrap navigation menus in `<nav>` elements with unordered lists. This enables assistive technologies to build a correct outline of your page [1].
- **Accessible names and labels**: Every interactive element needs an accessible name. Provide `aria-label`, `aria-labelledby`, or visible headings for dialogs (`role="dialog"`) [2] and `aria-label` for navigation menus. Use `aria-expanded` and `aria-controls` on accordion buttons [3].
- **Descriptive iframe titles**: When embedding video, set the `title` attribute on `<iframe>` elements so screen reader users know what the video contains.
- **Regions and landmarks**: Mark expanded accordion panels with `role="region"` and `aria-labelledby` referencing the controlling button [4]. Use `<main>` and `<nav>` landmarks to orient assistive technology users.
- **Focus management and traps**: Ensure that keyboard focus is moved into modals and remains trapped until the dialog closes. Return focus to the initiating element on close [5]. Provide visible focus styles, as in the global CSS, and ensure they meet a 3:1 contrast ratio.
- **Keyboard navigation**: Support `Tab`, `Shift+Tab`, `Enter`, and `Space` for all interactive controls [6]. Provide fallback for arrow keys if you adopt additional controls.
- **Color contrast**: Text and icons must meet WCAG color contrast guidelines—4.5:1 for normal text (AA) and 7:1 for enhanced contrast (AAA). Use a contrast checker when selecting colors [7].
- **Motion and animation**: Respect the user's `prefers-reduced-motion` preference by skipping or shortening animations. Provide a mechanism to pause or skip non-essential animations such as preloaders [8].
- **Skip links**: Include a "Skip to main content" link at the top of the page to allow keyboard users to bypass repetitive navigation.
- **Alternative text and descriptive content**: Provide `alt` attributes on decorative images (empty `alt=""`) and meaningful descriptions on informative images. For video content, supply captions or a transcript to support deaf and hard-of-hearing users.
- **Responsive and zoom friendly**: Ensure that layouts reflow gracefully at increased text sizes or when zoomed. Avoid fixed heights that truncate content; use relative units (`vh`, `svh`) and `min-height` where appropriate.
- **Test with assistive technologies**: Validate your implementation using screen readers (e.g. NVDA, VoiceOver) and keyboard-only navigation. Check that announcements are clear and that focus order is logical.

By following this checklist, the interactive features described in this PRD should deliver a native-feeling, accessible experience for all users, including those with visual, cognitive, and motor impairments.

---

[1] [4] [6] Accordion Example | APG | WAI | W3C
https://www.w3.org/WAI/ARIA/apg/patterns/accordion/examples/accordion/

[2] [5] Mastering Accessible Modals with ARIA and Keyboard Navigation - The A11Y Collective
https://www.a11y-collective.com/blog/modal-accessibility/

[3] [7] How to Code Accessible Accordions - Equalize Digital
https://equalizedigital.com/how-to-code-accessible-accordions/

[8] Understanding Success Criterion 2.2.2: Pause, Stop, Hide | WAI | W3C
https://www.w3.org/WAI/WCAG21/Understanding/pause-stop-hide.html