System Architecture Specification

Project: **image-carousel-adv** (React + Vite)

Generated: March 01, 2026

This document describes the architecture of the provided React application, including component responsibilities, data flow, runtime behavior, and key diagrams (component, class, and sequence).

Table of Contents

1. Executive Summary

The application renders a coverflow-style image carousel. On load, it fetches a list of images from the public Picsum API, stores them in React state, and lets the user navigate forward/backward through the list. The carousel effect is achieved by computing per-slide inline styles (transform/opacity/z-index/brightness) from each slide's offset relative to the current index, combined with CSS transitions for smooth animation.

The codebase is intentionally small: a top-level **App** component wires configuration, and a reusable **Carousel** component owns fetching, navigation state, rendering, and animation styling.

2. Goals, Non-Goals, and Constraints

• **Goal:** Fetch and display a finite list of images from a remote API with graceful loading and error states.

• **Goal:** Provide smooth "gliding" navigation (Prev/Next) with wrap-around behavior.

• **Goal:** Keep component API simple (url + limit) and reusable.

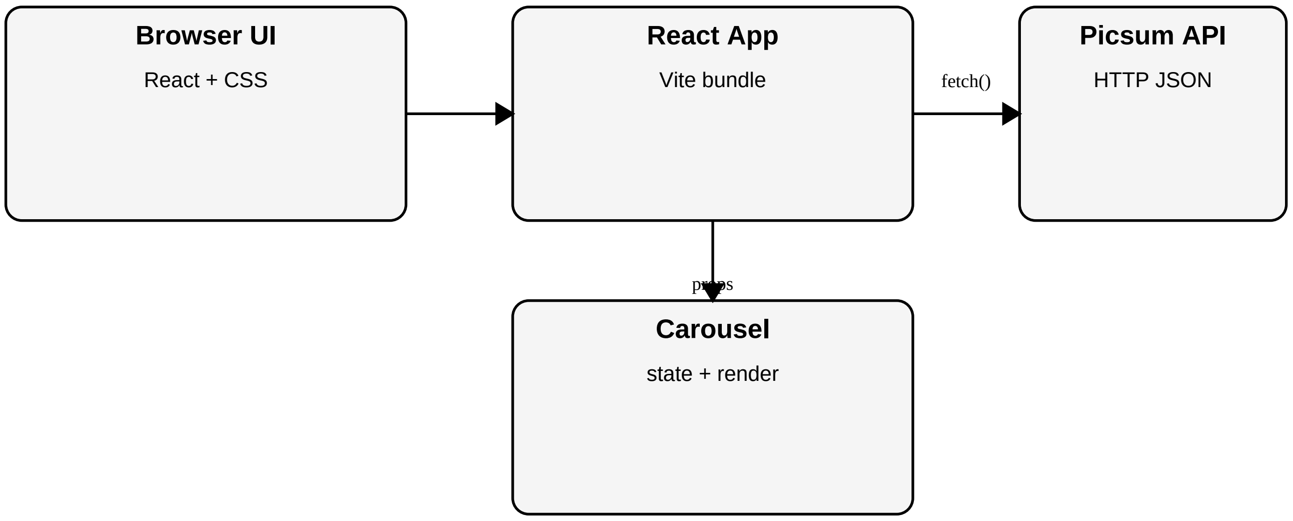
• **Constraint:** Client-only app (no backend service in this repo).

• **Constraint:** Animation uses CSS transitions + inline styles; no external animation libraries.

3. High-Level Architecture

At runtime, the React bundle executes in the browser. The Carousel performs an HTTP fetch to the external Picsum endpoint, then renders images using CSS positioning and transform transitions.

3.1 Component Diagram



Key interactions: App passes configuration via props; Carousel fetches JSON and converts it into rendered slides; user events update state which triggers re-render and CSS-driven animation.

4. Data Model

The carousel consumes a list of image objects returned by the Picsum endpoint. The code relies on the following fields:

Field	Type	Used for
id	string	Stable React key (prevents remounting; enables animation)
author	string	Alt text and caption
download_url	string (URL)	Image src for <img>

If the upstream API changes shape, the Carousel should introduce an adapter layer (mapping API JSON -> internal model) and validate required fields before rendering.

5. Component & Module Design

5.1 React Entry Points

The app is bundled with Vite. The main entry mounts **App** into the DOM.

```
import { StrictMode } from 'react' import { createRoot } from 'react-dom/client'
import './index.css' import App from './App.jsx'
createRoot(document.getElementById('root')).render( <StrictMode> <App />
</StrictMode>, )
```

5.2 App Component (src/App.jsx)

Responsibilities:

• Owns high-level configuration (Picsum endpoint, image limit).

• Renders page-level layout and passes props to Carousel.

• Keeps the Carousel reusable by not embedding fetch logic here.

```
import './App.css'; import Carousel from './components/Carousel'; function App() {
const PICSUM_API_URL = 'https://picsum.photos/v2/list'; return ( <> <h1>Image
Carousel</h1> <div className="card"> <Carousel url={PICSUM_API_URL} limit={15} />
</div> <p className="read-the-docs"> Click the arrows to glide through the images.
</p> </> ); } export default App;
```

5.3 Carousel Component (src/components/Carousel.jsx)

Responsibilities:

• Fetches image list from *url* using **useEffect** (re-runs when url/limit changes).

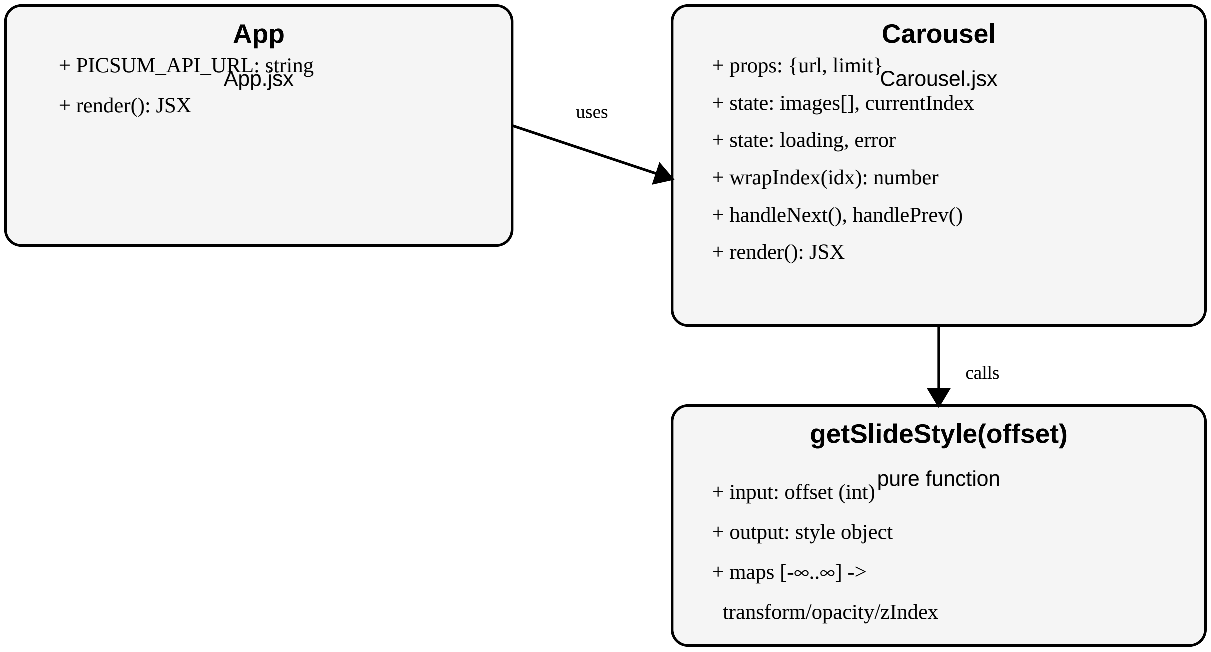
• Maintains UI state: images, currentIndex, loading, error.

• Implements cyclic navigation with wrap-around via wrapIndex().

• Computes per-slide visual placement using **offset** and getSlideStyle().

• Renders accessible controls (aria-label buttons, aria-live viewport).

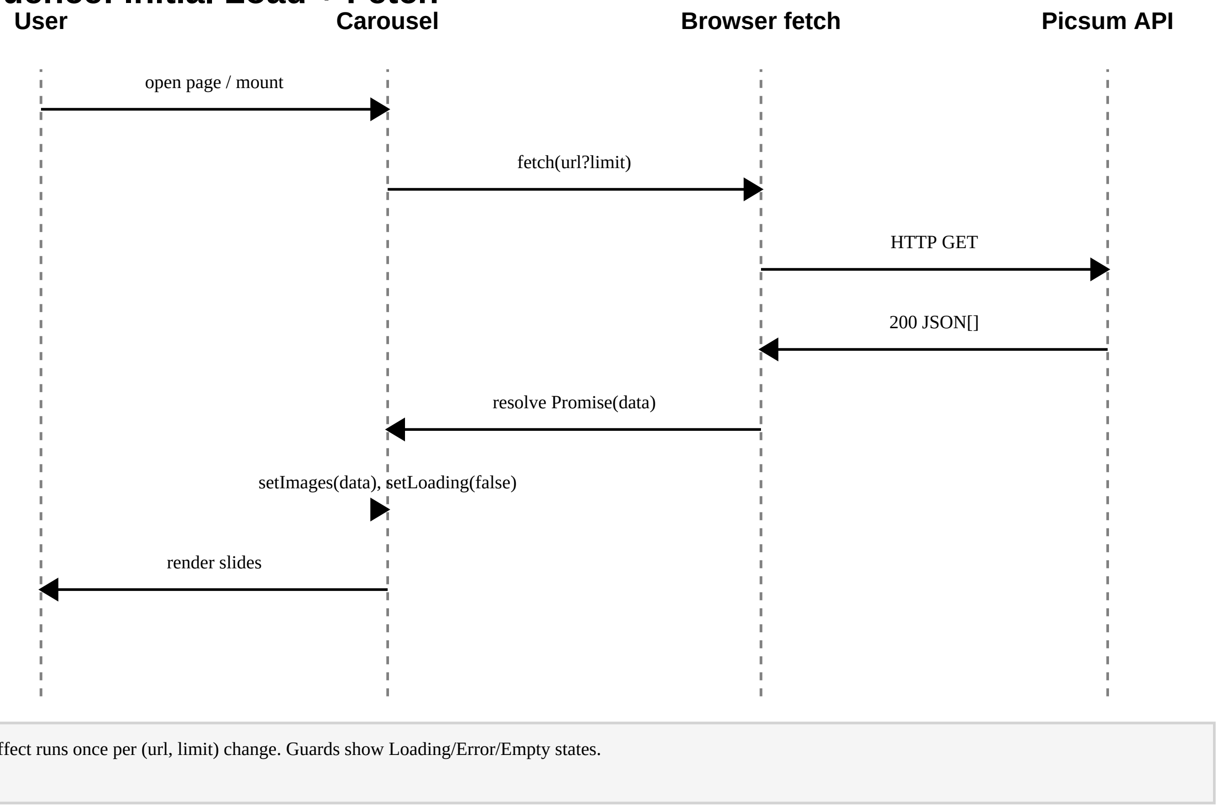
5.4 Class Diagram (Component Structure)



Although React is functional, the diagram treats modules as “classes” to clarify responsibilities and public surface area.

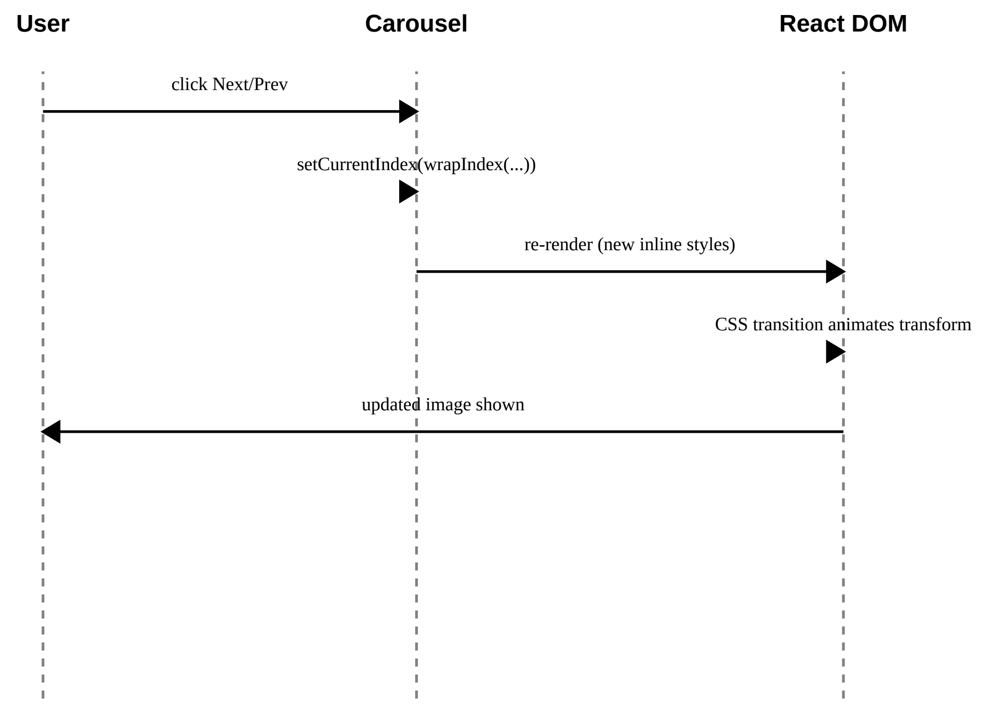
6. Runtime Behavior

6.1 Sequence: Initial Load + Fetch



During loading, the component shows a dedicated status element. If fetch fails (non-2xx or network error), the error is stored in state and rendered in a styled error banner.

6.2 Sequence: Next/Prev Navigation + Animation


User
Carousel
React DOM
click Next/Prev
setCurrentIndex(wrapIndex(...))
re-render (new inline styles)
CSS transition animates transform
updated image shown

Key insight: slides keep stable keys (image.id), so style updates animate instead of remounting.

7. Animation & Layout Strategy

The “coverflow” effect is produced by a deterministic mapping from relative slide offset -> CSS transform and visual attributes. The mapping lives in a pure helper function (**getSlideStyle**).

```
const getSlideStyle = (offset) => { const abs = Math.abs(offset); const sign = offset
> 0 ? 1 : -1; switch (abs) { /* ■■ Center (active)
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ */ case 0: return { transform:
'translate(-50%, -50%) scale(1.3)', opacity: 1, zIndex: 10, filter: 'brightness(1)',
boxShadow: '0 24px 60px rgba(0, 0, 0, 0.75)', }; /* ■■ Adjacent (±1)
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ */ case 1: /* * translateX moves the
card to the side. * translateY(-43%) instead of (-50%) nudges the card * downward by
~7% of slide height, so only the top portion * overl aps behind the center card —
reinforcing the "behind" feel. * scale(0.60) makes it noticeably smaller. *
brightness(0.50) darkens it so it reads as receded. */ return { transform:
`translate(calc(-50% + ${sign * 150}px), -50%) scale(0.80)`, opacity: 0.78, zIndex:
5, filter: 'brightness(0.40)', boxShadow: '0 8px 24px rgba(0, 0, 0, 0.55)', }; /* ■■
Outer edge (±2) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ */ case 2: return {
transform: `translate(calc(-50% + ${sign * 250}px), -50%) scale(0.50)`, opacity:
0.38, zIndex: 2, filter: 'brightness(0.20)', boxShadow: '0 4px 10px rgba(0, 0, 0,
0.40)', }; /* ■■ Hidden (|offset| > 2) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ */
default: /* * Kept in the DOM so they can animate OUT smoothly * (opacity:0 still
lets the transition play). * pointerEvents:none prevents them from intercepting
clicks. */ return { transform: `translate(calc(-50% + ${sign * 480}px), -50%)
scale(0.25)`, opacity: 0, zIndex: 0, filter: 'brightness(0)', pointerEvents: 'none',
}; } }; /* ■
```

Design choices that matter:

• **Stable keys:** slides use key=image.id, so DOM nodes persist and transitions animate style changes instead of remounting.

• **Shortest-path offsets:** offsets are normalized so wrap-around (last->first) animates as a one-step move.

• **Progressive de-emphasis:** opacity/brightness/scale decrease with distance from center; distant slides remain mounted for smooth transitions out.

8. Error Handling, Resilience, and Accessibility

• **Network failures:** caught in fetchImages(); rendered with distinct error styling.

• **Empty payload:** guards render “No images found.” if the API returns an empty list.

• **Keyboard & screen reader:** buttons include aria-label; viewport uses aria-live='polite' to announce changes.

• **Performance:** center image loads eagerly; others are lazy-loaded. Off-screen slides are visually hidden (opacity 0) and non-interactive (pointer-events none).

Recommended improvements for production: add AbortController to cancel in-flight fetch on unmount, add retry/backoff for transient failures, and support keyboard arrow navigation + focus management for improved accessibility.

9. Build, Run, and Deployment

The repository is a standard Vite React project. Key scripts:

Command	Purpose
npm run dev	vite
npm run build	vite build
npm run lint	eslint .
npm run preview	vite preview

Deployment options: static hosting (Netlify/Vercel/S3+CloudFront). Since the app is fully client-side, ensure the host serves index.html for the root path.

10. Extension Points

• **Replace data source:** keep Carousel API (url/limit) and swap endpoint; optionally add a mapper for different response schemas.

• **Add thumbnails:** render a strip of clickable thumbnails mapped to indices.

• **Add autoplay:** timer that triggers handleNext() with pause-on-hover.

• **Add gestures:** pointer/touch events for swipe navigation on mobile.

• **Add tests:** unit tests for wrapIndex + offset normalization; component tests for loading/error rendering.

End of document.