A Seminar Report

On

FRAMER MOTION

as the partial fulfillment of

Semester - VI study

for the degree of

Bachelor of Science (Information Technology)

(Affiliated to Sarvajanik University)

during the academic year 2024-2025.

Developed By:

Hunny Parmar

Internal Guide:

Mrs. Charmy Patel

Sarvajanik University

Shree Ramkrishna Institute of Computer Education and Applied Sciences

Behind P.T Science College, M.T.B. College Campus, Athwalines, Surat-395 001

Sarvajanik Education Society

Shree Ramkrishna Institute of Computer Education and Applied Sciences

Behind P.T. Science College, M.T.B College Campus, Athwalines, Surat-395 001.

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE

This is to certify that ***Hunny Parmar*** *with Exam No: SR22BSIT129 has successfully completed her seminar work entitled*

Framer Motion

as the partial fulfillment of

Semester -VI study

for the degree of

Bachelor of Science (Information Technology)

(Affiliated to Sarvajanik University)

during the academic year 2024-2025.

Internal Guide	**HOD** **Dept. of Computer Science**
Date: ____________	**Place: Surat**

University Examination:

Examination No. : ____________

Signature of External Examiners : ______________________________

INDEX

FRAMER MOTION

➢ Foundations of Framer Motion

Framer Motion is a declarative animation library for React that allows developers to create complex animations with ease. Before understanding Framer Motion itself, it's important to appreciate the evolution of animation on the web and why motion plays a vital role in user experience (UX).

Evolution of Web Animation

In the early days of the internet, web animations were crude and often relied on plugins like Flash. As web technologies matured, CSS animations and transitions became the norm. These allowed for simple animations using keyframes and transitions, but had limitations in terms of control and responsiveness.

JavaScript-based animation libraries like jQuery Animate, Anime.js, and GSAP gave developers more power and control over animations, allowing for timelines, sequencing, and easing. However, these were mostly imperative and introduced performance or maintainability challenges.

React introduced a new paradigm in UI development — declarative interfaces. This gave rise to animation tools like React Spring and ultimately, **Framer Motion**, which combines the power of animation with the declarative nature of React components.

Importance of Motion in UI/UX

Motion plays a critical role in improving user experience:

- **Feedback**: Motion helps users understand when an action is recognized (e.g., button tap).
- **Navigation**: Page transitions guide users through flows.
- **Hierarchy**: Animations can emphasize which elements are more important.
- **Delight**: Thoughtful animations can create emotional engagement.

Modern applications are expected to be fluid and interactive. Tools like Framer Motion are built to meet these expectations — offering smooth, efficient, and highly customizable animation capabilities.

➢ What is Framer Motion?

Framer Motion is an open-source motion library developed by the team behind Framer — a design tool for interactive prototyping. It is designed specifically for use with React and focuses on declarative animations, intuitive APIs, and fine-grained control.

Originally, Framer was a prototyping environment. But with the shift toward component-based development, the Framer team introduced **Framer Motion** to bring the same design-first motion capabilities directly into React applications.

Key Features of Framer Motion:

- **Declarative Syntax**: Easily readable and maintainable.
- **Variants**: Define complex animation states and reuse them across components.
- **Gesture Support**: Built-in support for hover, tap, drag, and more.
- **Shared Layout Transitions**: Animate between different layouts with ease.
- **AnimatePresence**: Handle component entry/exit animations in a declarative way.

Framer Motion fits seamlessly into the React ecosystem, leveraging hooks and component-based architecture.

React-Only vs Standalone Framer

Framer Motion is built exclusively for React and is not usable outside of it. In contrast, **Framer (the design tool)** supports design and prototype development independently. Framer Motion targets developers building production-ready interfaces, not just prototypes.

This makes Framer Motion perfect for teams where designers use Framer and developers implement the actual frontend using React and Framer Motion for consistent animated behaviors.

➢ Setup and Core Concepts

To begin using Framer Motion, developers need a React environment. The library can be installed via NPM and configured in just a few lines.

Installation

~ ***npm install framer-motion***

Once installed, you can import motion components into your project:

~ ***import { motion } from 'framer-motion';***

Creating Your First Animation

With Framer Motion, you can animate any element by wrapping it with a motion component. For example:

```
<motion.div
  initial={{ opacity: 0 }}
  animate={{ opacity: 1 }}
  transition={{ duration: 1 }}
>
  Hello, World!
</motion.div>
```

This simple example fades in the div from opacity 0 to 1 in one second.

Core Concepts

- **motion components**: Extended React elements that accept animation props.
- **initial**: The starting state before animation.
- **animate**: The target state to animate to.
- **exit**: The state to animate to when the component leaves the DOM.
- **transition**: Defines how the animation occurs (duration, delay, type).

These properties allow you to construct complex, layered animations with minimal code.

➢ Animation Techniques

Framer Motion supports advanced animation patterns beyond simple transitions. This includes reusable state-based animations, keyframes, gesture-triggered animations, and more.

Variants

Variants allow you to define multiple visual states and switch between them declaratively. This is useful when managing component animation states based on props or interaction.

```
const boxVariants = {
  hidden: { opacity: 0, scale: 0.8 },
  visible: { opacity: 1, scale: 1, transition: { duration: 0.5 } },
};
<motion.div
  variants={boxVariants}
  initial="hidden"
  animate="visible"
/>
```

Transitions

Transitions define how animations progress from one state to another. Common types:

- **Tween**: Default animation type using linear interpolation.
- **Spring**: Simulates physical motion with stiffness and damping.
- **Inertia**: Animates with momentum (useful in drag).

```
<motion.div
  animate={{ x: 100 }}
  transition={{ type: 'spring', stiffness: 300 }}
/>
```

Keyframes

For more control, Framer Motion supports keyframe-based animations:

```
<motion.div
  animate={{ scale: [1, 1.5, 1] }}
  transition={{ duration: 1 }}
/>
```

This scales the element up and down in a looping keyframe motion.

➢ Hooks and Triggers

Beyond declarative animations, Framer Motion provides hooks and event-based triggers that offer imperative control. These are useful in scenarios where animation behavior depends on user input or external conditions.

useAnimation Hook

The useAnimation hook allows you to start, stop, and control animations programmatically.

```
const controls = useAnimation();

<motion.div animate={controls} />

<button onClick={() => controls.start({ x: 100 })}>
  Move Right
</button>
```

This gives fine-grained control over motion, ideal for chained animations or conditionally triggered transitions.

whileHover / whileTap / onTap

You can also use built-in event handlers to trigger animation directly from user interaction:

```
<motion.button
  whileHover={{ scale: 1.2 }}
  whileTap={{ scale: 0.9 }}
>
  Click Me
</motion.button>
```

These interactions make it easy to add responsive, interactive elements without managing complex state.

➢ Layout and Presence Animations

Framer Motion makes layout transitions seamless by enabling components to animate between layout states using `layout` and `layoutId` props.

Layout Animations

By adding the layout prop to a motion component, you allow it to smoothly animate from one layout position to another:

<motion.div layout>...</motion.div>

Useful for animating between grid and list views, expanding sections, and more.

Shared Layout Transitions

With layoutId, multiple components can share a visual identity across transitions:

<motion.div layoutId="card">...</motion.div>

This is powerful for modal transitions, image gallery zoom-ins, or switching views while maintaining continuity.

Real-World Example

In a gallery app, clicking an image can transition it into a full-screen modal using the same layoutId. The animation feels natural, reducing cognitive load on users.

➢ AnimatePresence

Normally, when components unmount in React, they are immediately removed from the DOM. `AnimatePresence` allows you to animate components **out** before they're removed.

Setup

Wrap the animated element inside <AnimatePresence> and define the exit prop.

```
import { AnimatePresence, motion } from 'framer-motion';

<AnimatePresence>
  {isVisible && (
    <motion.div
      initial={{ opacity: 0 }}
      animate={{ opacity: 1 }}
      exit={{ opacity: 0 }}
    >
      Goodbye!
    </motion.div>
  )}
</AnimatePresence>
```

This is essential for conditional rendering, modals, toast messages, dropdowns, etc.

exitBeforeEnter

This prop ensures that a component fully exits before another enters.

```
<AnimatePresence exitBeforeEnter>...</AnimatePresence>
```

Combining AnimatePresence with layout transitions can yield dynamic, immersive user flows.

➢ Motion Values and Performance

Framer Motion provides powerful tools to track animation state and optimize performance.

MotionValue

motionValue tracks real-time values like x, opacity, scale:

```
const x = useMotionValue(0);
<motion.div style={{ x }} />
```

It can be combined with useTransform to derive new values:

```
const scale = useTransform(x, [0, 100], [1, 2]);
```

This allows for chaining effects and creating reactive animations.

useMotionValue Event Listeners

You can subscribe to motion value updates:

```
x.onChange((latest) => console.log(latest));
```

This can be used to synchronize animations with scroll, gestures, or sensor inputs.

LazyMotion

By default, Framer Motion imports the entire animation engine. To optimize bundle size, use LazyMotion:

```
import { LazyMotion, domAnimation } from 'framer-motion';

<LazyMotion features={domAnimation}>
  <motion.div ... />
</LazyMotion>
```

LazyMotion loads features on demand, reducing initial load times in large apps.

➢ Framework Integration

Framer Motion integrates well with popular frontend frameworks and UI libraries.

React Integration

Framer Motion is built specifically for React. It embraces the component lifecycle and React hooks for seamless animations. You can directly animate React components by wrapping them with motion.

```
import { motion } from 'framer-motion';

const AnimatedBox = () => (
  <motion.div initial={{ opacity: 0 }} animate={{ opacity: 1 }}>
    Hello World
  </motion.div>
);
```

This integration is natural for developers already comfortable with React. Animation states like initial, animate, exit, and transition can be passed as props, making the animation logic predictable and declarative.

Tailwind CSS, Chakra UI

You can combine Framer Motion with utility-first CSS libraries like Tailwind or component-based systems like Chakra UI:

```
<motion.div className="bg-blue-500 p-4" animate={{ scale: 1.1 }} />
```

Or wrap Chakra components with motion:

```
const MotionBox = motion(Box);
```

This enables expressive UIs with motion and design tokens.

➢ Accessibility and Testing

Animations should always consider accessibility needs and testing processes.

Reduced Motion

1. Use useReducedMotion() Hook in React: The useReducedMotion() hook detects whether the user has set their system to reduce motion, allowing you to adjust or disable animations accordingly.

```
const shouldReduceMotion = useReducedMotion();
```

Adjust animations accordingly to reduce distraction or dizziness for motion-sensitive users.

2. CSS Media Queries: You can also use the CSS prefers-reduced-motion media query to respect system preferences. This media query allows you to target users who prefer less motion and disable animations or apply less intense effects

```
@media (prefers-reduced-motion: reduce) {
  * {
    animation: none !important;
  }
}
```

Important Note: By using animation: none, you can ensure that all animations, including scrolling, transitions, and keyframe animations, are disabled, reducing any potential discomfort for motion-sensitive users.

Testing Animations

Testing animations is critical to ensure they function as expected, do not break functionality, and remain consistent across different environments and browsers. There are several approaches to testing animations:

1. **Unit Testing with Jest and React Testing Library:**

Unit tests ensure that individual components behave as expected. Although Jest doesn't natively support testing animations, you can still check if the component properly handles animation state changes (e.g., whether an animation has started or finished based on state).

For example, using React Testing Library, you can test if a component renders in an expected state before and after an animation:

```
import { render, fireEvent } from '@testing-library/react';
import MyComponent from './MyComponent';

test('should start animation when button is clicked', () => {
  const { getByTestId } = render(<MyComponent />);
  const button = getByTestId('animate-button');

  fireEvent.click(button);
  const animatedElement = getByTestId('animated-element');
  expect(animatedElement).toHaveClass('is-animated'); // Check if animation class is added
});
```

This test simulates a button click that triggers an animation and verifies that the animation is applied by checking for the appropriate class or state change.

2. **Visual Regression Testing:**

Visual regression testing tools like **Percy** or **Chromatic** are essential to ensure that animations look consistent across deployments. These tools can capture snapshots of your UI over time and compare them to previous snapshots, highlighting any visual differences.

For animations, visual regression testing ensures that:

- The timing and appearance of animations remain consistent across different browsers and screen sizes.
- The layout and behavior of elements remain intact after updates or code changes.

These tools capture screenshots of your app or website before and after changes, and compare them pixel by pixel, detecting unintended visual changes.

Example: You might want to test that the animated transition of a dropdown menu opens smoothly and stays consistent even after code updates or browser changes. Percy or Chromatic will help ensure this visual consistency.

3. **Behavioral Testing:**

Testing animations is not just about visuals; it's also about ensuring that animations behave as expected in different conditions:

- **Performance:** Ensure that animations do not negatively impact performance. Test on low-end devices to verify that animations are smooth and do not cause jank or performance issues.
- **Accessibility:** Check if the reduce-motion preferences work correctly across different browsers and devices, ensuring that the animations are properly disabled when needed.

Best Practices for Animation Testing

- **Test across devices:**

  Animations may look different on various devices (e.g., mobile vs. desktop) due to differences in hardware and browser rendering engines. Make sure to test your animations on a variety of devices.

- **Check performance:**

  Use browser tools like Chrome DevTools to check for dropped frames or performance bottlenecks during animations. Optimize animations to run smoothly by using transform and opacity over properties like top or left to reduce layout recalculations.

- **Test timing:**

  Verify that animations start and finish at the correct times, especially when chained or triggered by user actions. You can mock timers and use tools like Jest to check if timeouts and intervals are being respected during tests.

➢ Evaluation & Library Comparison

Framer Motion has quickly established itself as the preferred animation library for React developers. Its declarative syntax, out-of-the-box gesture support, layout animations, and ease of integration with other UI libraries contribute to its widespread adoption. Below, we provide an in-depth analysis of its advantages and disadvantages and how it compares with other animation libraries.

Advantages of Framer Motion

- **Declarative and Intuitive**:

  Framer Motion offers a syntax that is easy to read and write. Developers can animate components using simple props like animate, initial, and exit.

- **Built for React**:

  Unlike many generic libraries, Framer Motion is designed specifically for React. It takes advantage of hooks and the component model.

- **Variants and State-Driven Animations**:

  Animation logic is easy to encapsulate in variants, which can be reused across multiple components.

- **Gestures and Interaction**:

  Built-in support for whileHover, whileTap, drag, and focus make it easy to create interactive UIs without additional tools.

- **Layout and Shared Layout Animations**:

  Framer Motion supports fluid layout transitions using layout and layoutId, allowing for advanced UI effects like shared element transitions.

- **Presence Animations**:

  With AnimatePresence, it becomes easy to animate components when they are removed from the DOM.

- **Performance Features**:

  Features like LazyMotion and motionValue ensure that animations are GPU-accelerated and efficient.

Disadvantages of Framer Motion

- **React-Only**:

  Framer Motion works only with React. Developers using Vue, Angular, or plain JavaScript need other alternatives.

- **Bundle Size**:

  While reasonably optimized, Framer Motion is larger in size compared to some smaller, single-purpose libraries.

- **Advanced Logic Can Be Complex**:

  While simple animations are easy, chaining animations with useAnimation() or coordinating complex sequences can become verbose.

Comparison with Other Animation Libraries

1. Framer Motion vs GSAP (GreenSock Animation Platform)

- **GSAP Strengths**:
  - Superior timeline control for sequencing.
  - Extremely performant.
  - Used widely in large-scale and high-performance animation projects.
- **Framer Motion Strengths**:
  - Simpler, more React-friendly.
  - Built-in support for gestures and layout animations.
  - Ideal for UIs and component transitions.

Verdict: Use GSAP for custom visual storytelling, but Framer Motion is better for React apps and interactive UIs.

2. Framer Motion vs React Spring

- **React Spring Strengths**:
  - Physics-based animations using spring dynamics.
  - Supports declarative and imperative styles.
- **Framer Motion Strengths**:
  - Easier to use and more consistent.
  - Richer feature set out-of-the-box (e.g., exit animations, variants).

Verdict: React Spring is great for fluid, physics-based motion; Framer Motion is preferred for structured animation logic in larger applications.

3. Framer Motion vs Anime.js

- **Anime.js Strengths**:
  - Powerful and flexible imperative animations.
  - Good support for SVG, DOM, and CSS animations.
- **Framer Motion Strengths**:
  - Declarative and React-based.
  - Easier state synchronization and integration with component lifecycle.

Verdict: Anime.js is suited for one-off animations or non-React projects; Framer Motion is ideal for React UIs.

➢Use Cases of Framer Motion

Framer Motion's versatility has made it a core part of UI design in modern applications. Here are some of its most compelling use cases across industries and product types:

1. Product Landing Pages

Motion is critical in the first impression of any website. Landing pages leverage scroll-triggered animations, text reveals, and interactive hero sections to guide the user's attention and highlight key offerings.

2. Mobile Web Interfaces

With built-in support for drag and gesture animations, Framer Motion is highly effective in simulating native-like interactions on mobile. Examples include draggable cards, swipe-to-dismiss components, and animated bottom sheets.

3. Dashboards and Admin Panels

Framer Motion adds clarity and delight to data-heavy interfaces. Animated filtering, expanding panels, and smooth chart transitions all improve user experience and interaction depth.

4. Modals, Alerts, and Overlays

Using AnimatePresence and layoutId, modals can be animated in and out with beautiful transitions that reinforce flow and context. Toasts and notifications also benefit from natural fade or slide-in effects.

5. Interactive Forms and Wizards

Multi-step form transitions feel intuitive when pages slide or fade during user progress. Framer Motion helps break down complex processes into digestible chunks.

6. Media Viewers and Galleries

For applications showing videos, image grids, or audio libraries, Framer Motion offers zoom animations, gallery modal popups, and hover-triggered previews that make the content feel dynamic and rich.

7. Educational Apps and eLearning Platforms

Animated components increase engagement and retention. Framer Motion aids in animating concepts, explanations, and user feedback without disrupting interactivity.

Framer Motion’s declarative approach and ease of use allow developers to focus on interaction design without reinventing complex motion behaviors. The result: faster development, consistent UX, and elevated digital experiences.

➢ Best Practices & Tips

When working with Framer Motion, there are several best practices to keep in mind for maintainable, performant code.

1. Keep Animation Declarative

Framer Motion encourages a declarative approach. By using props like initial, animate, and exit, you ensure that your animations are clear, maintainable, and tied directly to component lifecycles.

```
<motion.div initial={{ opacity: 0 }} animate={{ opacity: 1 }} exit={{ opacity: 0 }} />
```

Avoid mixing imperative logic unless necessary, as this can lead to harder-to-maintain codebases.

2. Use Variants for Reusability

Variants allow you to define reusable animation states. They promote a consistent animation structure, especially in component hierarchies.

```
const item = {
  hidden: { opacity: 0 },
  visible: { opacity: 1 }
};
```

Use them to manage groups of items or toggle animation states easily.

3. Combine useMotionValue with useTransform

These hooks allow you to build reactive animations based on user interaction or layout changes.

```
const x = useMotionValue(0);
const rotate = useTransform(x, [-100, 100], [-45, 45]);
```

This setup is perfect for scroll effects, sliders, and parallax transitions.

4. Optimize with LazyMotion

For large-scale apps, LazyMotion helps split out animation features and reduces the initial bundle size.

```
import { LazyMotion, domAnimation } from 'framer-motion';
<LazyMotion features={domAnimation}>
  <motion.div animate={{ scale: 1.2 }} />
</LazyMotion>
```

This can significantly enhance performance on initial load.

5. Respect Accessibility

Always account for users with reduced motion preferences. Use the useReducedMotion() hook to disable or simplify complex animations.

```
const shouldReduceMotion = useReducedMotion();
const animation = shouldReduceMotion ? { opacity: 1 } : { opacity: [0, 1] };
```

This ensures that your application remains inclusive.

6. Test in Context

Animations should reinforce your application’s interactions—not hinder them. Use real user flows to test if your animations:

- Feel natural
- Enhance clarity
- Do not impact usability

Tools like Storybook and Chromatic can also help test and preview animations across UI states.

7. Avoid Overuse of Motion

Use motion where it serves a purpose—drawing attention, providing feedback, or guiding users. Too much animation can distract and overwhelm the user.

8. Structuring Animation Logic

- Keep animation logic close to components.
- Use custom hooks (useToggleAnimation) for complex flows.
- Document motion behavior alongside component docs.

9. Performance Optimization

- Avoid animating layout-breaking properties like width, height, or top. Prefer transform and opacity.
- Profile animations using DevTools Timeline to check frame rates.

10. Responsive Motion Design

- Adjust animation timing and behavior for mobile vs desktop.
- Keep gestures consistent across input types (touch/mouse).

➢Future Scope of Declarative Animation

The future of declarative animation is poised for exciting transformations, with Framer Motion at the helm of innovation. As user interfaces evolve, the need for scalable, intuitive, and maintainable animation systems becomes even more critical. Below are key trends and directions that will define the future of declarative animation:

Component-Driven Animation Systems

Declarative animation thrives in component-centric architectures. Framer Motion integrates motion directly into React components, allowing UI logic and motion behavior to coexist. This pattern is increasingly being adopted across frontend ecosystems, leading to tighter coupling of behavior and visuals.

Future frameworks will likely expand on this principle, enabling more advanced motion patterns without leaving the component structure. Such systems will make motion an inherent aspect of reusable UI components.

Integration with Design Tools

The boundary between design and development continues to shrink. Framer, as a design tool, already supports code-export features. As this tooling matures, developers may soon work with motion specifications directly sourced from design software like Figma or Framer.

Declarative animations will play a key role in making this integration seamless—allowing animations to be defined once and interpreted across tools and environments.

<u>Web + Native Motion Consistency</u>

With the rise of tools like React Native Reanimated and Expo Motion, there's a push toward unifying the web and native animation paradigms. Declarative approaches promise cross-platform consistency, reducing the learning curve and increasing code reuse between web and mobile platforms.

Libraries inspired by Framer Motion’s philosophy may appear in other ecosystems, bringing component-first, variant-driven motion to broader audiences.

<u>AI-Powered Motion Design</u>

As AI becomes more involved in UI development, future animation libraries might harness machine learning to suggest, generate, or even adapt motion patterns based on user behavior, device usage, or content structure.

For instance, a smart system might slow down animations on low-powered devices, simplify them for accessibility, or enhance them for high engagement zones—all automatically and declaratively.

<u>Composability and Expressiveness</u>

Framer Motion has set a high bar with composable features like variants, layout animations, and motion values. As the demand for dynamic interfaces grows, the need for composability and expressive APIs will become standard.

Animation libraries will need to provide declarative ways to define complex behavior chains, transitions, and state-based logic without cluttering application code.

➢ Conclusion

In this documentation, we have explored the power and flexibility of **Framer Motion** in creating dynamic and visually engaging animations within React applications. From basic animations to complex transitions, Framer Motion offers an intuitive API that empowers developers to create smooth, interactive, and responsive user interfaces with ease.

By leveraging key features such as motion components, animation variants, gestures, and layout animations, Framer Motion enhances the user experience and adds an extra layer of polish to React projects. We also discussed important topics like performance optimization, accessibility considerations, and testing, ensuring that animations are not only visually appealing but also functional, inclusive, and efficient.

Incorporating Framer Motion into your projects can lead to more interactive, delightful user experiences while maintaining ease of use, flexibility, and scalability. Whether you're building simple UI animations or complex, choreographed transitions, Framer Motion provides the necessary tools to bring your animations to life seamlessly.

➢ References

Framer Motion Documentation - Framer Motion Docs

Official documentation for Framer Motion, covering installation, basic usage, and advanced features.

React Documentation - React Docs

The official React documentation for understanding component structure, hooks, and other essential features.

MDN Web Docs on CSS Animations - MDN CSS Animations

A comprehensive guide to CSS animations, useful for understanding animation properties in the context of web development.

React Spring Documentation - React Spring Docs

Another popular animation library for React, providing an alternative to Framer Motion for more physics-based animations.

Visual Regression Testing with Percy - Percy

Tool for visual regression testing that captures snapshots and compares them to ensure consistency.

Jest Testing Framework - Jest Docs

Official Jest documentation for testing JavaScript applications, including unit and snapshot testing.