College Code : 9607

College Name : Immanuel Arasar JJ College Of Engineering

Department :CSE

NM-Id : au960701

Roll No :960723104001

Date :25-09-2025

Completed The Project Named As Phase: R.Ahalya,S.Rajkumar,S.Bharath,R.Sunilkumar,T.Hari krishnan

Technology Project Name :Dynamic Image Slider

Submited By

Name:R.Ahalya

Mobile no:9843479548

DYNAMIC IMAGE SLIDES

ENCHACEMENT&DEPLOYMENT

CONTENTS:

Additional Features:

1. **Captions and Descriptions:**

- Show text overlays on every slide (heading, details, links).
- Can animate the text in and out when each slide transitions.

Great for storytelling or emphasizing products.

2. Video Slides:

- Allow videos to be embedded (like YouTube or MP4) in the slides.
- Videos can autoplay or pause when the slide changes.

Combine images and videos smoothly.

3. Thumbnail Preview on Hover:

- When you hover over the navigation dots or arrows, display a small thumbnail preview of the next slide.

4. Keyboard Navigation:

- Let users use the arrow keys on the keyboard (← and →) for navigation.

5. Multiple Sliders on One Page:

- Allow your slider to be used multiple times on the same page, with each having its own controls and state.

6. Progress Bar or Timer Indicator:

- Display a visual bar or circular timer that fills up as the slide moves to the next one.

7. Full-screen Mode:

- Include a button to switch to fullscreen mode for viewing the slider.

8. Shuffle / Random Slide Order:

- Incorporate a shuffle feature to present images in a random sequence.

9. Loop Control:

- Provide an option for users to enable or disable endless looping.

10. Thumbnail Carousel Navigation:

- Include a scrollable row of thumbnails either underneath or next to the main slider for easy access.

11. Zoom and Pan:

- Allow users to zoom in on images and navigate around the enlarged view.

12. Lazy Load with Placeholder:

- Display a blurred temporary image or a loading spinner while the main images are being loaded.

13. Social Sharing Buttons:

- Facilitate the direct sharing of the current slide or image on social media networks.

14. Touch Gestures Enhancements:

- Introduce pinch-to-zoom, double-tap for zooming, and speed detection for swiping to improve mobile user experience.

15. Autoplay Customization:

- Give users the ability to adjust autoplay speed or switch autoplay on or off.

UI/UX Improvements:

1. Fluid Animations and Transitions:

- Incorporate easing functions such as ease-in-out or cubic-bezier to achieve smoother sliding or fading visuals.
- Introduce gentle scaling or opacity changes on hover for buttons and indicators.
- Implement animations for caption texts using fade, slide-in, or zoom effects.

2. User-Friendly Controls:

- Enlarge navigation arrows and place them comfortably with sufficient padding.
- Expand the clickable areas of buttons and indicators to improve accessibility.
- Offer visual responses like highlighting buttons or scaling them upon click or tap.

3. Clear Progress Indicators:

- Utilize progress bars or countdown timers to indicate how long each slide will remain visible.
- Clearly emphasize the active dot by using a contrasting hue or increasing its size.

4. Flexible and Adjustable Layout:

- Make certain the slider adjusts fluidly across various screen dimensions.
- Modify navigation controls, such as smaller arrows on mobile devices or hiding dots if there's too much clutter.
- Adopt adaptable font sizes and margins that respond to viewport width.

5. Stop on Interaction:

- Suspend autoplay upon hover, focus, or swipe to empower users and prevent unexpected slide transitions.

6. Accessibility for Keyboards and Screen Readers:

- Enable navigation through the keyboard using arrow keys and tab focus.
- Incorporate ARIA labels and roles for controls and slides.
- Ensure that screen readers effectively announce changes in slides and content meaningfully.

7. Visual Harmony and Theming:

- Maintain a consistent use of colors, shadows, and typography that align with your branding.
- Apply soft shadows or overlays on slides to enhance text legibility.
- Consider supporting both dark and light mode.

8. User Notifications and Loading States:

- Display loading spinners or placeholders prior to the appearance of images.
- Give responses for button clicks or swipe actions.
- Utilize skeleton loaders or blur-up effects for a progressive loading of images.

9. Simple and Clean Design

- Eliminate clutter by keeping controls straightforward and user-friendly.
- Utilize whitespace effectively to create a spacious feel for the slider.

10. Touch-Friendly Controls

- Ensure that controls are large enough and adequately spaced for touch use.
- Include sensitivity adjustments for swipe gestures to create a natural touch experience.

API Enhancement:

1. Filtering & Query Parameters

- Permit users to narrow down the slides they wish to see.

Examples:

`GET /api/slides?category=travel`

`GET /api/slides?lang=en`

`GET /api/slides?active=true&limit=5`

Benefits:

- Display different sliders based on various pages, users, or regions.
- Support pagination or limits for better efficiency.

2. Multi-language Support (i18n):

Configure the API to provide localized titles and descriptions for the slides.

JSON Example:

```json
{
"image": "https://example.com/image.jpg",
"title": {
"en": "Welcome to the Jungle",
"fr": "Bienvenue dans la Jungle"
},
"description": {
"en": "Explore the untamed wild.",
"fr": "Explorez la nature sauvage."
```

}

}

API Usage:

`GET /api/slides?lang=fr`

Your frontend would retrieve `title[fr]`.

---

3. Scheduling & Expiration:

- Include `start_date` and `end_date` fields to only show currently relevant slides.

JSON Example:

```json
{
"image": "slide1.jpg",
"start_date": "2025-09-01",
"end_date": "2025-10-01"
}
```

API Logic:

- Ensure only to return slides where:

```js
```

```
start_date <= today && (end_date === null || end_date >= today)
```

4. Slide Types (Image / Video / CTA):

- Enable slides to be more dynamic by including a `type` field.

JSON Example:

```json
{
"type": "video",
"video_url": "https://youtu.be/xyz123",
"thumbnail": "thumb.jpg"
}
```

Types You May Support:

`image`

`video`

`cta` (Call to Action)

`product`

5. Personalized/Authenticated Slides:

- Allow users to access specific slides using a token or user ID.

API Call:

```http
GET /api/slides?user_id=12345
Authorization: Bearer {token}
```

Example Use Case:

- Users who are logged in can view their saved promotions.
- Content targeted by location based on user profile.

6. Ordering & Priority:

- Introduce an `order` field or `priority_score` to arrange how slides appear.

```json
{
"order": 1,
"priority_score": 92.5
}
```

7. Pagination / Infinite Scroll:

- Improve performance when there are numerous slides.

API Example:

```http
GET /api/slides?page=1&limit=5
```

Response:

```json
{
"page": 1,
"total": 25,
"per_page": 5,
```

"data": [...]

}

8. Status Control:

- Control slide visibility with `status` values such as:

```json

"status": "draft" // or "active", "archived"

**API Filters**:

`GET /api/slides?status=active`

**9. CMS Integration:**

- Support your API with a CMS such as:

Strapi (self-hosted, REST/GraphQL, customizable)

Sanity (provides real-time updates)

Contentful or Prismic

WordPress REST API

- These options allow content managers a user interface for editing slides without needing developer assistance.

**10. Security Enhancements:**

If your slider contains private or personalized content:

- Require user authentication (JWT, API keys).
- Implement rate limiting (e.g., 100 requests per hour).
```

- Configure CORS to limit which frontend domains can access it.

11. CDN Optimization:

- Serve optimized URLs from a CDN, ensuring appropriate sizes for quicker loading times.

```json
{
"image": "https://cdn.example.com/images/slide1.webp?w=800&auto=format"
}
```

12. GraphQL Alternative:

- For more complex query requirements, shift to using a GraphQL API:

```graphql
{
slides(limit: 5, lang: "en", status: "active") {
title
description
image
videoUrl
}
}
```

Performance and Security Checks:

Performance Evaluations:

1. Image Enhancement:

- Utilize compressed image formats such as WebP, AVIF, or optimized JPEG/PNG.
- Provide various sizes suited for different devices through srcset.
- Employ a Content Delivery Network (CDN) such as Cloudflare, Cloudinary, or Imgix for caching and resizing purposes.

2. Deferred Loading:

- Only load the current slide and the adjacent ones when the page first opens.
- Implement loading="lazy" in   elements or apply lazy load techniques in JavaScript.

3. Compressed Files:

- Compress HTML, CSS, and JavaScript files for production use.
- Combine and remove unused JavaScript using tools like Webpack, Rollup, or Vite.

4. Effective API Requests:

- Store the API response in memory or local storage if the slider content remains stable.
- Limit or delay API requests when retrieving personalized or paginated

information.

5. Segmentation or Virtual Rendering:

- If your slider consists of numerous items:
- Employ API pagination (with limit and offset) and load slides as necessary.
- Think about "virtual rendering" — only display slides that are currently visible in the DOM.

6. Browser Caching & HTTP Settings:

- On your API or server:
- Configure Cache-Control, ETag, and Expires headers.
- Utilize HTTP/2 or HTTP/3 for quicker multiplexed requests.

7. Code Division:

- Load solely what is necessary:
- Include slider JavaScript and CSS only on the pages where they are required.
- Use dynamic import() for optional modules like video players.

Testing of Enchancements:

Now we will create a modal class for storing our URLs for our images. For creating a new java class. Navigate to the app > java > your app's package name and Right-click on it and click on New > Java Class. Give a name to your java class and add the below code to it. Here we have given the name as SliderData. Below is the code for the SliderData.java file.

```
public class SliderData {

    // string for our image url.
    private String imgUrl;

    // empty constructor which is
    // required when using Firebase.
    public SliderData() {
    }

    // Constructor
    public SliderData(String imgUrl) {
        this.imgUrl = imgUrl;
    }

    // Getter method.
    public String getImgUrl() {
        return imgUrl;
    }

    // Setter method.
    public void setImgUrl(String imgUrl) {
        this.imgUrl = imgUrl;
    }
}
```

- Evaluating improvements for dynamic image slides usually requires following various best practices, based on the platform and the complexity of the functionalities.
- Essential elements to evaluate comprise the navigation sequence (including next/previous buttons), seamless transitions, responsiveness, and the dynamic retrieval or refresh of images from external data sources.

Deployment(Netlify,Vercel,or Cloud Platform):

1. **Netlify:**

   Ideal for static websites (simple HTML/JS, Angular, Vue, and React).

   Link the Bitbucket, GitHub, and GitLab repositories.

   Deploy automatically with every git push.

   In Site Settings → Build & Deploy, set up environment variables.

   supports form handling and serverless functions.

Vercel:

- Excellent for frontend frameworks like React and Next.js.
- One-click connection with Bitbucket, GitHub, and GitLab.

- Build and deploy automatically with each push.

  integrated serverless features.

- Project Settings → Environment Variables is where you'll find

  AWS, GCP, and Azure are examples of cloud platforms.
- Ideal for full-stack or large-scale applications.

- For static hosting, use CloudFront (AWS) in conjunction with S3.

- For front-end and back-end development, use Firebase Hosting (GCP).

- For straightforward projects, use Azure Static Web Apps.

- CI/CD pipelines through Jenkins, GitLab CI, or GitHub Actions.

Gitgub: