College Code : 9607

College Name : Immanuel Arasar JJ College Of Engineering

Department :CSE

NM-Id :au960701

Roll No :960723104001

Date :23-09-2025

Completed The Project Named As Phase:

R.Ahalya,S.Rajkumar,S.Bharath,R.Sunilkumar,T.Harikrishnan

Technology Project Name :Dynamic Image Slider

Submited By

Name:R.Ahalya

Mobile no:9843479548

DYNAMIC IMAGE SLIDES

MVP IMPLEMENTATION

Contents:

- Project Setup
- Core Features Implementation
- Data Storage(Local State/Data base)
- Testing Core Features
- Version Control(Gitgub)

Introduction:

In an MVP implementation, the goal is to build a basic,working version of dynamic image slide with minimal complexity but enough functionality to deliver value and test with users.Dynamic image slides are a popular feature used to display multiple images in smooth,automated,and interactive way on website or applications.They enchance user experience by presenting visual content dynamically,rather than as static images.

Project setup:

1. Establish the project's objective :

   Construct a carousel or image slider with dynamically changing images. Support both automated sliding and manual navigation (next/prev). Make it simple to add fresh pictures.

   2.Folder Structure:

   The file ```dynamic-image-slides/ │ ├── index.html → Primary entry page ├── style.css → Layout and transition styling ├── script.js → Logic for loading and sliding dynamic images └── assets/ → (Optional) Save local photos ```

   3. Important Elements

   HTML file

   Slider container Navigation buttons Dynamically loaded image placeholder

   CSS file

   Button and arrow styling; Animation effects (fade/slide transitions); Layout (width, height, alignment)

A file in JavaScript

Manage the next or previous slide Auto-slide with timer Responsive behavior (mobile-friendly) Load photos (from array, API, or folder).

Core Features Implementation:

1. Dynamic Source Image Loading:

Retrieve or import pictures from picture URLs in a static array.

folder for local assets.

JSON and remote API (for completely dynamic loading).

Dynamically add images to the slider container.

2. Navigation on Slides

Next Button → Advances to the following picture.

Previous Button → Takes you back to the earlier picture.

Looping → Return to the first image if the user reaches the last one (and vice versa).

3. Sliding Automatically

Set a timer to advance slides automatically, for example, every three to five seconds.

In order to prevent conflicts, reset the timer if the user clicks next or previous.

Looping indefinitely (auto restarts after the final image).

4. Effects of Transition:

Slides can be smoothly animated (e.g., fade, slide, zoom).

JavaScript-controlled animations or CSS transitions.

Keep your time constant (e.g., 0.5s).

5. Indicators (essential to contemporary sliders, but optional):

Thumbnails or tiny dots beneath the slider.

Draw attention to the slide indicator that is now active.

Select the indicator to go straight to that slide.

6. Adaptability:

The slider adapts to various screen sizes:

Mobile full width.

7. Availability:

Keyboard navigation.

For next/previous, use the arrow keys.

8. Stop and Start:

Auto-slide is paused when:Over the slider, the user hovers.

The user engages with the navigation.

After the interaction, resume the auto-slide.

Data Storage(Local State/Data Base):

1. Local State:

Use cases include prototypes, static websites, and small projects.

- How it operates:

Store picture URLs in a JavaScript state variable or array.

An example of a structure

[{ "url": "assets/image1.jpg", "caption": "First Slide" }, { "url": "assets/image2.jpg", "caption": "Second Slide" }, { "url": "assets/image3.jpg", "caption": "Third Slide" }]

Advantages:

Easy, quick, and server-free.

2. IndexedDB (Browser Storage) :

Use case:

Sliders that allow users to customize them (e.g., uploading own photos).

How it operates:

Store picture data in IndexedDB (big data, such as Base64 images) or localStorage (little data).

Retrieve the cached images when the page loads.

Advantages:

Persists between sessions and operates offline.

3.Database equipped with a backend API:

Use case:

CMS-driven sliders for many users (e.g., portfolios, ecommerce banners).

Options:

Structured data (picture URL, caption, order, tags) -> SQL (MySQL, PostgreSQL).

NoSQL → flexible schema (excellent for JSON picture objects) (MongoDB, Firebase).

How it operates:

Save the metadata (URL, alt text, caption, and order).

Data is served to the frontend as JSON by APIs (Node.js/Express, Django, etc.).

Advantages:

Dynamic changes that are centralized and don't involve programming.

4. Database + Cloud Storage:

Use case:

Big programs that require actual picture hosting.

Setup:

Keep real photos in a cloud storage bucket (Firebase Storage, Google Cloud Storage, or AWS S3).

Keep image metadata in a database, including URLs, captions, and order.

5. Ready-to-use Headless :

Use case:

Content managers who don't know how to code can update slider images.

Strapi, Contentful, Sanity, and WordPress (headless) are a few examples.

How it operates:

Image assets and metadata are stored by CMS.

Frontend retrieval over REST/GraphQL API.

Advantages:

Updating photographs is simple for non-technical people.

Testing Core Features:

1. Dynamic Source Image Loading:

Do all photos come from the array, folder, or API and are loaded dynamically?

Do broken URLs (like placeholder images) get handled gracefully?

Test Procedures:

The slider page should load.

Verify that every image is shown in the slider.

Replace a URL with an invalid link for a short time to watch how the slider responds.

Anticipated Outcome:

All legitimate photographs show up, and mistakes are handled politely.

2. Next/Previous Slide Navigation:

Things to check:

The next image is displayed when you click Next.

To view the previous image, click Previous.

Looping works by going back to the first image after the last one (and vice versa).

Test Procedures:

Repeatedly click the Next button to confirm the order.

Several times, click the Previous button → Check the opposite

4. Effects of Transition:

Things to check:

Slide transitions, such as fade and slide, are seamless.

No sudden jerks or flickers.

Test Procedures:

Keep an eye out for changes while auto-slidding.

To view the animations, click Next/Previous.

Anticipated Outcome:

Consistent, seamless slide transitions.

5. Dots or Indicators (Optional):

Things to check:

The active slide is appropriately highlighted.

A dot click takes you to the appropriate slide.

Test Procedures:

Watch the active dot as the slides change on their own.

Click each dot to confirm that the correct slide is displayed.

Anticipated Outcome:

Slides navigate and indicators update accurately.

6.Adaptability:

Things to check:

The slider adjusts to several screen sizes, including desktop, tablet, and mobile.

Pictures keep their aspect ratio.

Test Procedures:

Use a device emulator or resize the browser window.

Take note of the button positioning, image scaling, and layout.

Anticipated Outcome:

The slider exhibits complete responsiveness and visual consistency.

7. Availability:

Things to check:

You can use the keyboard to access the buttons (tab + Enter/Space).

Alt text is present in images.

Test Procedures:

Slider navigation is done only with the keyboard.

Verify each image's alt text (for screen readers).

Anticipated Outcome:

Compatible with screen readers and keyboard users.

8. Stop and Start:

Things to check:

When the user hovers over the slider, auto-slide pauses.

After the interaction is over, auto-slide restarts.

Test Procedures:

Verify that the auto-slide stops by hovering over the slider.

Move the mouse aside to confirm that auto-slides are resumed.

Anticipated Outcome:

The pause/resume behavior functions as planned.

Version Control(Gitgub):

For handling dynamic image sliders, version control—more especially, Git and sites like GitHub—is quite helpful since it makes it possible to track changes, collaborate, and publish the system quickly.

How to Create a Dynamic Image Slider with GitHub:

Configuring the Repository:

Make a new image slider project repository on GitHub.
Create a README.md file to serve as the project's initial description.

Project Organization:

Arrange your files in the repository in a sensible manner.

The following could be a typical structure: index.html (for the slider's display).

For styling, use style.css.
script.js (for loading images and dynamic behavior) .
images/ (a directory where you can keep your photos).
data/ (optional; used to store image metadata; images.json, for example).

Workflow for Version Control:

Make a copy of the repository: Code
Clone using git .

Modify:
Develop your JavaScript, HTML, CSS, and add/edit pictures.

Changes in stage: Code

Add git.
Make adjustments.
Code

Push the code to GitHub with git commit -m "Descriptive commit message"

Git push origin main Branches:
To prevent upsetting the main codebase, use branches for bug fixes or new features.

Code :

Make changes with git checkout -b new-feature #...
Git push origin new-feature git commit -m "Implemented new feature"

Merge:
Return a feature to the main branch after it has been finished and tested.

Code :

Git checkout main
Git merge new-feature
Git push origin main

Managing Images and Other Dynamic Content:

Image files should be kept in the repository: Include images directly in the images/directory and commit them to Git for smaller projects or when they are a component of the main application.

External storage for pictures that are large or change frequently:

Store photos outside (for example, in cloud storage like AWS S3 or Cloudinary) for larger projects or when they are controlled by a content management system (CMS). After that, your script.js would retrieve picture URLs from a version-controlled data file (such as images.json) or an API.

Gitgub: