UNIT IV Notes

Topic1: Importance of Layout(Purpose of Layout):

Layout plays a huge role in your website's page. Without it, your website would be a wall of text and links and no one would care to read through.

It improves user experience

A good layout improves user experience by creating a better, well, experience. Without a layout users can become frustrated when there's too much content on the screen making it overwhelming and stressful, resulting in someone just leaving the website.

It makes content more readable

When you put your content into a layout (instead of just making a giant wall of text) it makes it much more readable. For most pages (unlike blog posts) content isn't too text-dense making it able to be in a layout accompanied by images and other visual elements.

It creates smoother flowing pages

With a good layout, the content of a website flows a lot more smoothly creating a gradual informative experience. When your website flows better it creates a better reading experience for the user, promoting them to read your content a lot more.

Better ease of use

A good layout just makes websites easier to use. A better user experience, content readability, flow, predictability all work together to provide easier and smoother experiences for users.

It improves the design of your pages

To put it simply, pages look better with content in a layout rather than just spitting it out on the page. In blog posts it makes sense to not have much of a layout at all, but more top-level

important pages should have some sort of layout to make them stand out and make a good first impression.

It helps with user retention

With a layout there's more visual interest on the page keeping users on the page. A good layout is interesting enough to keep someone's attention but isn't too unfamiliar enough that the content becomes hard to consume and take in.

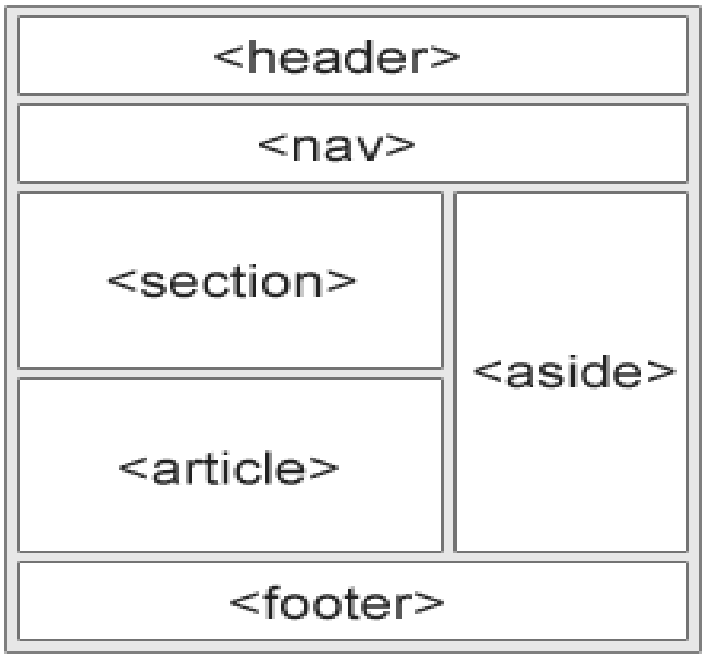
Branding:

A good website layout plays a role in branding too, using spacing, alignment and scale in ways that are consistent with the company's brand.

Topic 2: Explain in detail about HTML Layout Elements:

HTML Layout Elements

HTML has several semantic elements that define the different parts of a web page:



- ✔ **<header> -** Defines a header for a document or a section
- ✔ **<nav> -** Defines a set of navigation links
- ✔ **<section> -** Defines a section in a document
- ✔ **<article> -** Defines an independent, self-contained content
- ✔ **<aside> -** Defines content aside from the content (like a sidebar)
- ✔ **<footer> -** Defines a footer for a document or a section
- ✔ **<details> -** Defines additional details that the user can open and close on demand
- ✔ **<summary> -** Defines a heading for the <details> element

HTML <header> Element:

- ✔ The <header> element represents a container for introductory content or a set of navigational links.
- ✔ A <header> element contains:
  - ✔ one or more heading elements
  - ✔ logo or icon
  - ✔ authorship information

Example: A header section in a document:

```
<header>
  <h1>Knowledge Hub Company</h1>
  <p>Technology Center</p>
</header>
```

HTML <footer> Element

- ✔ The <footer> element defines a footer for a document or section.
- ✔ A <footer> element contains:

1. authorship information
2. copyright information
3. contact information
4. sitemap
5. back to top links
6. related documents

Example: A footer section in a document:

```
<footer>
  <p>Author: VSailaja</p>
  <img src=”address.jpg”>
</footer>
```

HTML <section> Element

- ✔ The <section> element defines a section in a document.
- ✔ "A section is a thematic grouping of content, typically with a heading."
- ✔ Examples of where a <section> element can be used:

1. Chapters
2. Introduction
3. News items
4. Contact information

Example: Creating Two different sections:

```
<html>
<body>

<section>
  <h1>WWF</h1>
  <p>The World Wide Fund for Nature (WWF) is an international organization working on issues regarding the conservation, research and restoration of the environment, formerly named the World Wildlife Fund. WWF was founded in 1961.</p>
</section>

<section>
  <h1>WWF's Panda symbol</h1>
  <p>The Panda has become the symbol of WWF. </p>
</section>

</body>
</html>
```

HTML <article> Element

- ✔ The <article> element specifies independent, self-contained content.
- ✔ An article should make sense on its own, and it should be possible to distribute it independently from the rest of the web site.

Examples of where the <article> element can be used:

1. Forum posts
2. Blog posts
3. User comments
4. Product cards
5. Newspaper articles

Example: Creating 2 different Articles

```
<html>
<body>
<article>
<p>Google Chrome is a web browser developed by Google, released in 2008. Chrome is the world's most popular web browser today!</p>
</article>

<article>
<p>Mozilla Firefox is an open-source web browser developed by Mozilla. Firefox has been the second most popular web browser since January, 2018.</p>
</article>
</body>
</html>
```

HTML <nav> Element

- ✔ The <nav> element defines a set of navigation links.
- ✔ We will use this to create menus of the webpage

Example: A set of navigation links:

```
<nav>
  <a href="Home.html">Home</a>
  <a href="Aboutus.html">About Us</a>
  <a href="Services.html">Services</a>
  <a href="Branches,html">Branches</a>
</nav>
```

HTML <aside> Element

The HTML <aside> tag is used to represent a portion of a document that is indirectly related to the main content. It is most commonly used as a sidebar in the document.

Example: Display some content aside from the content it is placed in:

```
<aside>
<p>  Aside tag is use to display important information about the primary page.</ p>
</aside>
```

Topic3: Explain about <details> and <summary>

HTML <details> Tag is used for the content/information which is initially hidden but could be displayed if the user wishes to see it. This tag is used to create an interactive widget that the user can *open or close.*

HTML <summary> tag defines a visible heading for the <details> element.
The heading can be clicked to view/hide the details.

Syntax:

```
<details>
   <summary>  ….  </summary>
    <p> Any Content we can place here</p>

 </details>
```

Note: The <summary> element should be the first child element of the <details> element.

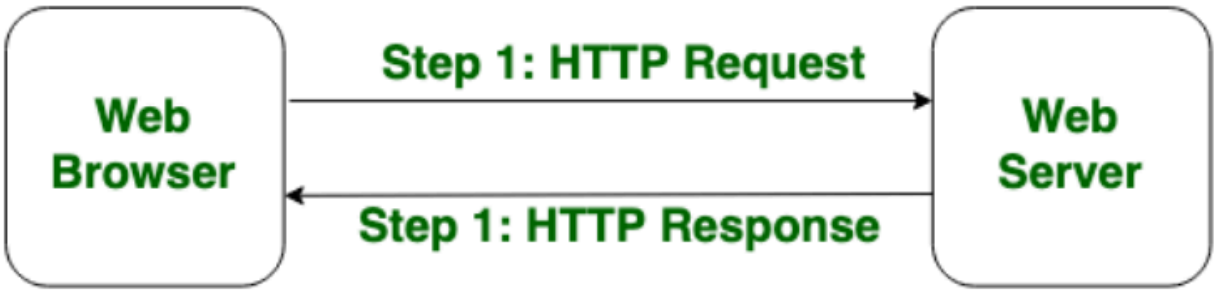
Program:

```
<html>
<body>


<details>
  <summary>Semester I </summary>
  <p>Subject1</p>
<p>Subject2</p>
<p>Subject3</p>
<p>Subject4</p>
<p>Subject5</p>
<p>Subject6</p>
</details>
<details>
  <summary>Semester II </summary>
  <p>Subject1</p>
<p>Subject2</p>
<p>Subject3</p>
<p>Subject4</p>
<p>Subject5</p>
<p>Subject6</p>
</details>

<details>
  <summary>Semester III </summary>
  <p>Subject1</p>
<p>Subject2</p>
```

```
<p>Subject3</p>
<p>Subject4</p>
<p>Subject5</p>
<p>Subject6</p>
</details>

<details>
  <summary>Semester IV </summary>
  <p>Subject1</p>
<p>Subject2</p>
<p>Subject3</p>
<p>Subject4</p>
<p>Subject5</p>
<p>Subject6</p>
</details>
</body>
</html>
```

Topic 4: Explain about Static Web pages:

Static Web pages are very simple. It is written in languages such as HTML, JavaScript, CSS, etc. For static web pages when a server receives a request for a web page, then the server sends the response to the client without doing any additional process. And these web pages are seen through a web browser. In static web pages, Pages will remain the same until someone changes it manually.



Static Web Page

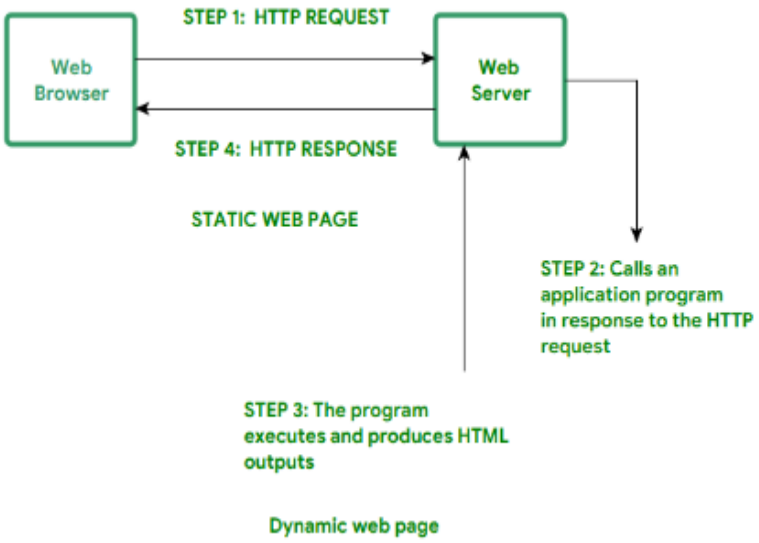
Program: You can write Any one program from Unit I

Topic 5: Explain about Dynamic Web Pages:

Dynamic websites are those that generate web pages in real-time. Web scripting code, such as PHP or ASP, is used on these pages. The Web server parses the code, and the generated HTML is transmitted to the client's browser when a client requests a dynamic page.

As dynamic websites are easier to maintain than static websites, the majority of large websites are dynamic. Static pages have their own set of content, hence they must be manually opened, changed, and published whenever they are updated.

Dynamic pages, on the other hand, use a database to store the data. As a result, the webmaster may need to edit a database record to change the content of a dynamic page. This is particularly useful for sites with hundreds or thousands of pages. It also allows numerous users to update a website's content without having



Program: You can any program from UNIT III (Any one program from UNIT III-Event Handling)

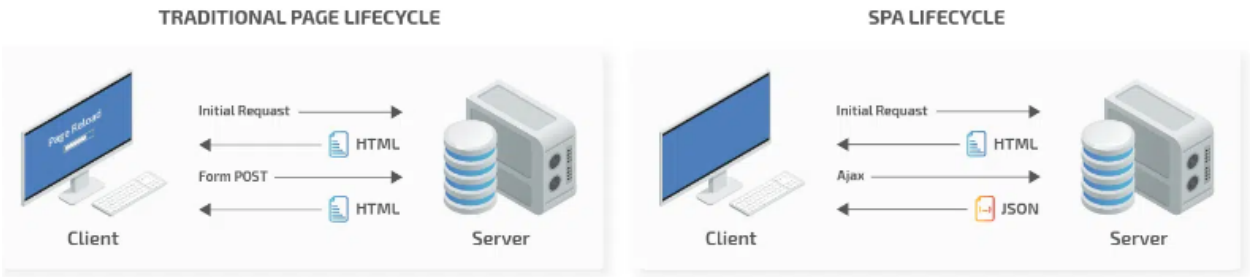
Difference between Static and Dynamic Web Pages:

S.NO	Static Web Page	Dynamic Web Page
1.	In static web pages, Pages will remain the same until someone changes it manually.	In dynamic web pages, Content of pages are different for different visitors.
2.	Static Web Pages are simple in terms of complexity.	Dynamic web pages are complicated.
3.	In static web pages, Information are change rarely.	In dynamic web page, Information are change frequently.
4.	Static Web Page takes less time for loading than dynamic web page.	Dynamic web page takes more time for loading.
5.	In Static Web Pages, database is not used.	In dynamic web pages, database is used.
6.	Static web pages are written in languages such as: HTML, JavaScript, CSS, etc.	Dynamic web pages are written in languages such as: CGI, AJAX, ASP, ASP.NET, etc.
7.	Static web pages does not contain any application program .	Dynamic web pages contains application program for different services.
8.	Static web pages require less work and cost in designing them.	Dynamic web pages require comparatively more work and cost in designing them.

Topic 6: Explain about Single Page Designs:

What is a single-page Designs(application)?

- A single-page application is a web app that consists of a single HTML page, loads new data from the server, and rewrites the page in response to user actions.
- Unlike Multi Page Applications(MPA), Single Page Applications(SPA) don't need to load each new web page from scratch — they simply alter the existing page.
- This approach allows for creating single-page applications that run in a browser.



Advantages of Single-Page Applications:

Fast response to user actions. To execute a user's request, an SPA loads a small JSON file instead of a new web page. Even with caching and lazy loading, MFAs take more time to get new data from a server than does an SPA. Also, the fact that the page doesn't need to reload makes workflows that involve multiple screens more streamlined in SPAs than in regular web applications.

High performance and stability. Asynchronous requests, when done right, can reduce server load and require less bandwidth. The core logic of a single-page application is usually fully downloaded at the first request, making it possible to perform most operations offline, providing a great solution for poor internet connections.

Fast development. During the single-page application development, the front end and back end can be separated, so two developers may work in parallel. Also, changing the front end doesn't affect the back end, and vice versa.

Desktop-like user experience. An SPA allows you to port on-premises applications to the web without making big changes to the UI or established workflow. This speeds up web development and provides users with a high-quality and familiar experience.

Mobile-friendly. An SPA back end can be used to create a mobile application identical to the web app. It doesn't require much adaptation, since an SPA is already designed as an application, not a website. This greatly speeds up mobile development.

Disadvantages of Single-Page Applications

Let's take a closer look at the drawbacks of this technology:

Lack of scalability. In regular web apps, functionality is spread across different pages, and it's easy to make changes to individual pages. SPA components are designed to fit together in a certain way, and any changes to a widely used component may have a big impact across the application. This is why SPAs need to be created with foresight about future changes, allowing developers to take them into account at the design stage.

Issues with architectural changes. If you need, for example, to switch the framework of a regular MPA, you can easily do it by changing one page at a time. SPAs don't allow for such incremental changes. You would need to port the entire application to a new framework at once. That's why developers have to be extremely careful when working on an SPA's architecture.

Poor search engine optimization (SEO). When a web crawler visits an SPA, it only indexes the main page and pays no attention to additional pages. Google is continuously updating its search algorithms to fix this issue, and SEO agencies have some workarounds to improve optimization. Still, SPAs aren't always aren't always indexed correctly.

Requires JavaScript. Some users disable JavaScript in their browsers to speed up loading. Since SPAs fully depend on JavaScript frameworks, they can't load in browsers with JavaScript disabled.

Memory leaks. Memory leaks within SPAs can be more severe than within multi-page applications, since the page is never fully reloaded but persists for a long time, which can slow down even fast machines. If an SPA is coded poorly, it can sometimes send even more requests to the server than an MPA.

Security considerations. With an SPA, clients download the whole application, which provides more opportunities for reverse engineering and finding vulnerabilities. You need to make sure that all single-page app security-related client-side logic (input validation, authentication, etc.) is doubled on the server for verification and that users are granted access only based on their roles regardless of the request.

Best frameworks for SPA development

Single page application examples

1. Netflix

An online entertaining and streaming platform Netflix is built on the React framework, so delays don't interfere with the user's viewing experience. With an SPA approach, Netflix can stream a huge quantity of data to simultaneous platform users. Netflix loads new data just in the browser when a viewer makes a new request, hence its speed.

2. Gmail
Gmail is a single-page application — meaning you don't need to reload the page when clicking the messages in your inbox. As all the data is loaded from the server at once. Same with Google Calendar and Drive.

3. Grammarly
The online writing assistant Grammarly checks your grammar and spelling in real time largely thanks to AI, because of the advanced functionality of SAP, and because it's built on a Vue.js framework. As a result, a user can receive Grammarly real-time suggestions directly on the Google Doc to make their writing clear, correct, and understandable.

4. Uber
As an SPA, Uber provides two user interface versions — one for drivers and one for passengers — in which both can see real-time updates such as points on a map, costs, account data, and driving time.

5. Spotify
As the user navigates Spotify, one of the world's largest SPA streaming music platforms, content loads dynamically and lets users choose, listen to, and search for favorite songs, playlists, and albums.

6. Airbnb
Similar to Spotify, Airbnb shares an unlimited number of apartments in any location. The functionality of SPA makes the rental experience of millions of users seamless and speedy.

SPAs became extremely popular because of active support from Google, which adopted this format for most of its services. Today, SPAs are mostly used for:

- social networks (Facebook, Twitter, LinkedIn)
- dashboards (Jira, Trello)
- web analogs of on-premises apps (Google Docs, Slack, Telegram)
- software as a service products
- dynamic projects that involve a small amount of data
- internal corporate projects.

Topic 7: Explain about Responsive Pages

- Responsive web design is about creating web pages that look good on all devices!
- A responsive web design will automatically adjust for different screen sizes and viewports.

What is Responsive Web Design?

Responsive Web Design is about using HTML and CSS to automatically resize, hide, shrink, or enlarge, a website, to make it look good on all devices (desktops, tablets, and phones):

Setting The Viewport:

To create a responsive website, add the following <meta> tag to all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This will set the viewport of your page, which will give the browser instructions on how to control the page's dimensions and scaling.

Responsive Images

Responsive images are images that scale nicely to fit any browser size.

Using the width Property

If the CSS width property is set to 100%, the image will be responsive and scale up and down:

```
<img src="img_girl.jpg" style="width:100%;">
```

Using the max-width Property

If the `max-width` property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size:

```
<img src="img_girl.jpg" style="max-width:100%;height:auto;">
```

Show Different Images Depending on Browser Width

The HTML `<picture>` element allows you to define different images for different browser window sizes.

Resize the browser window to see how the image below changes depending on the width:

```
<picture>
  <source srcset="img_smallflower.jpg" media="(max-width: 600px)">
  <source srcset="img_flowers.jpg" media="(max-width: 1500px)">
  <source srcset="flowers.jpg">
  <img src="img_smallflower.jpg" alt="Flowers">
</picture>
```

Responsive Text Size

The text size can be set with a "vw" unit, which means the "viewport width".That way the text size will follow the size of the browser window:

```
<h1 style="font-size:10vw">Hello World</h1>
```

Programs on responsive pages:

Program1:

```
<html>
<head>
<meta name="viewport" content="width=device-
width, initial-scale=1.0">
</head>
<body>

<h2>Setting the Viewport</h2>
<p>This example does not really do anything, other
than showing you how to add the viewport meta
element.</p>

</body>
</html>
```

Program2:

```
<html>

<head>
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
</head>
<body>

<h2>Responsive Image</h2>
<p>When the CSS width property is set in a percentage value, the
image will scale up and down when resizing the browser window. Resize
the browser window to see the effect.</p>

<img src="rose.jfif" style="width:100%">

</body>
</html>
```

Program3:

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
</head>
<body>

<h2>Responsive Image</h2>
<p>"max-width:100%" prevents the image from getting bigger than its
original size. However, if you make the browser window smaller, the
image will still scale down.</p>
<p>Resize the browser window to see the effect.</p>

<img src="rose.jfif" style="max-width:100%;height:auto;">

</body>
</html>
```

Program4:

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
</head>
<body>

<h2>Show Different Images Depending on Browser Width</h2>
<p>Resize the browser width and the image will change at
600px,800px and 1500px.</p>

<picture>
  <source srcset="rose1.jpg" media="(max-width: 600px)">
  <source srcset="rose2.jfif" media="(max-width: 1500px)">
  <source srcset="rose3.jpg">
  <img src="rose.jfif" alt="Flowers" style="width:auto;">
</picture>

</body>
</html>
```


Program5:

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
</head>
<body>

<h1 style="font-size:10vw;">Responsive Text</h1>

<p style="font-size:5vw;">Resize the browser window to see how the
text size scales.</p>

<p style="font-size:5vw;">Use the "vw" unit when sizing the text.
10vw will set the size to 10% of the viewport width.</p>

<p>Viewport is the browser window size. 1vw = 1% of viewport width.
If the viewport is 50cm wide, 1vw is 0.5cm.</p>

</body>
</html>
```