

Single Page Applications

Concepts and Implementation



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

Table of Contents

1. Web Application Concepts
2. JavaScript Modules
3. SPA Approaches
4. Live Demo



sli.do

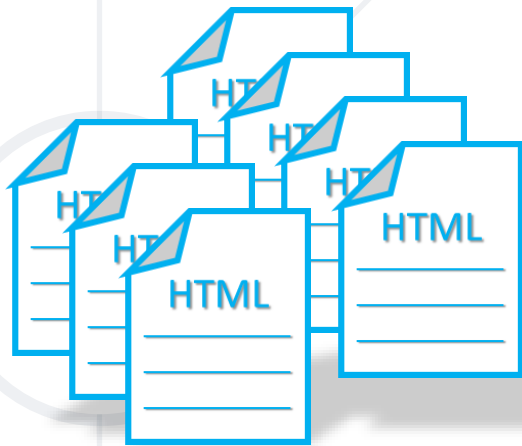
#js-advanced



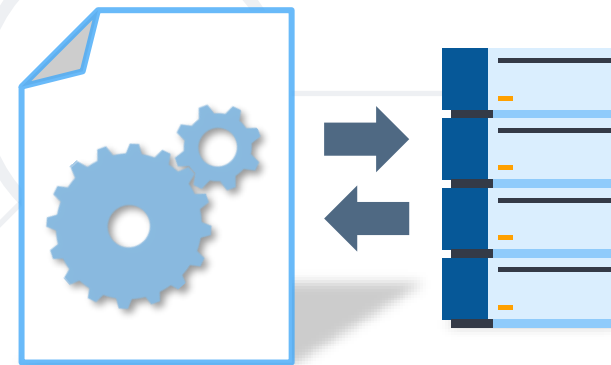
Web Application Concepts

Site vs Application. Multi- and Single-Page Apps

- A **website** is a collection of interlinked **web pages**
 - It hosts content, that is **primarily** meant to be **consumed**
- A **web application** is a **software**, accessible from a web browser
 - They are **interactive** and have rich **functionality**



Web Site



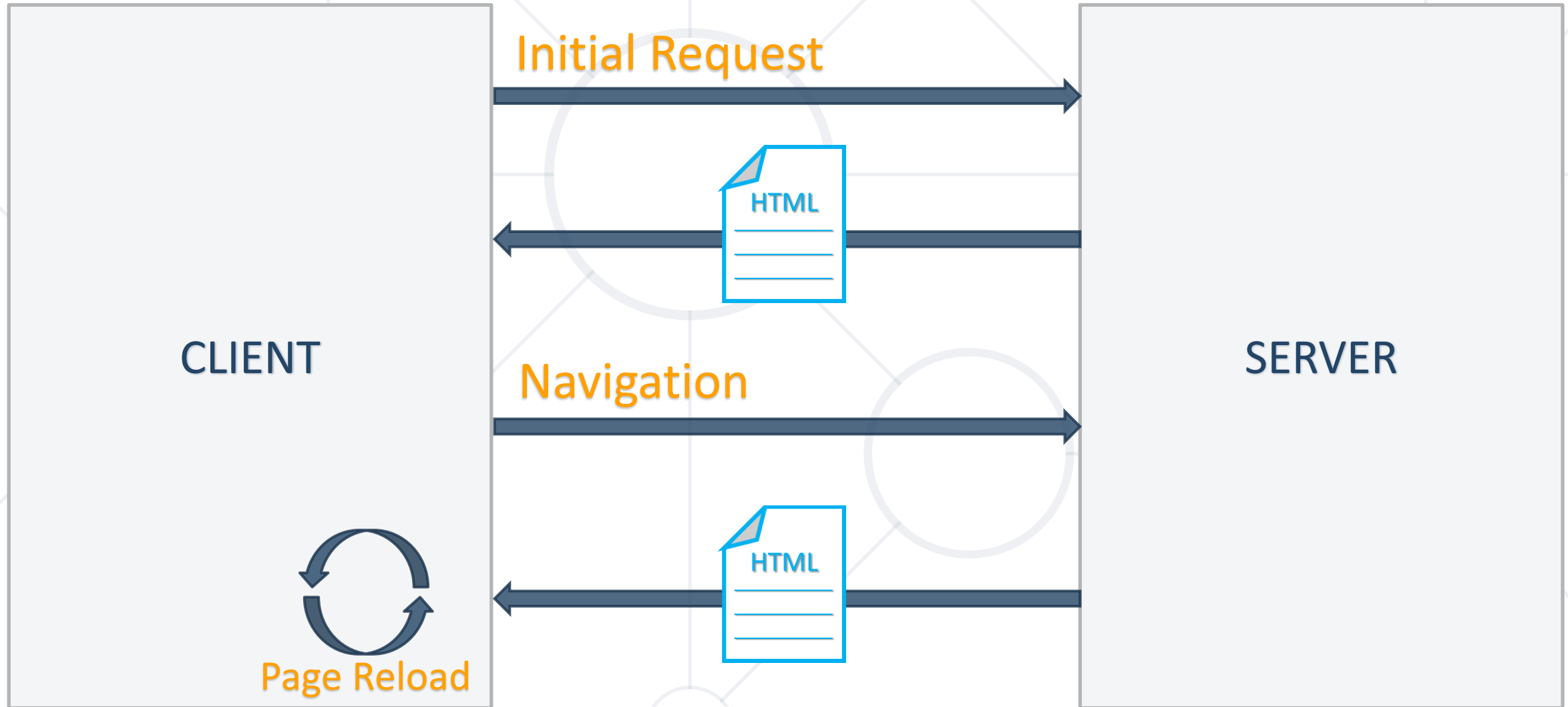
Web Application

Multi Page Applications

- **Reloads** the entire page
- **Displays** the **new page** when a user interacts with the web app
- When a data is exchanged, a **new page** is **requested** from the server to display in the web browser



Multi Page Application Lifecycle



Multi Page Pros and Cons

■ Pros

- Performs well on the **search engine**
- Provides a **visual map** of the web app to the user

■ Cons

- Comparatively **complex development**
- Coupled backend and frontend

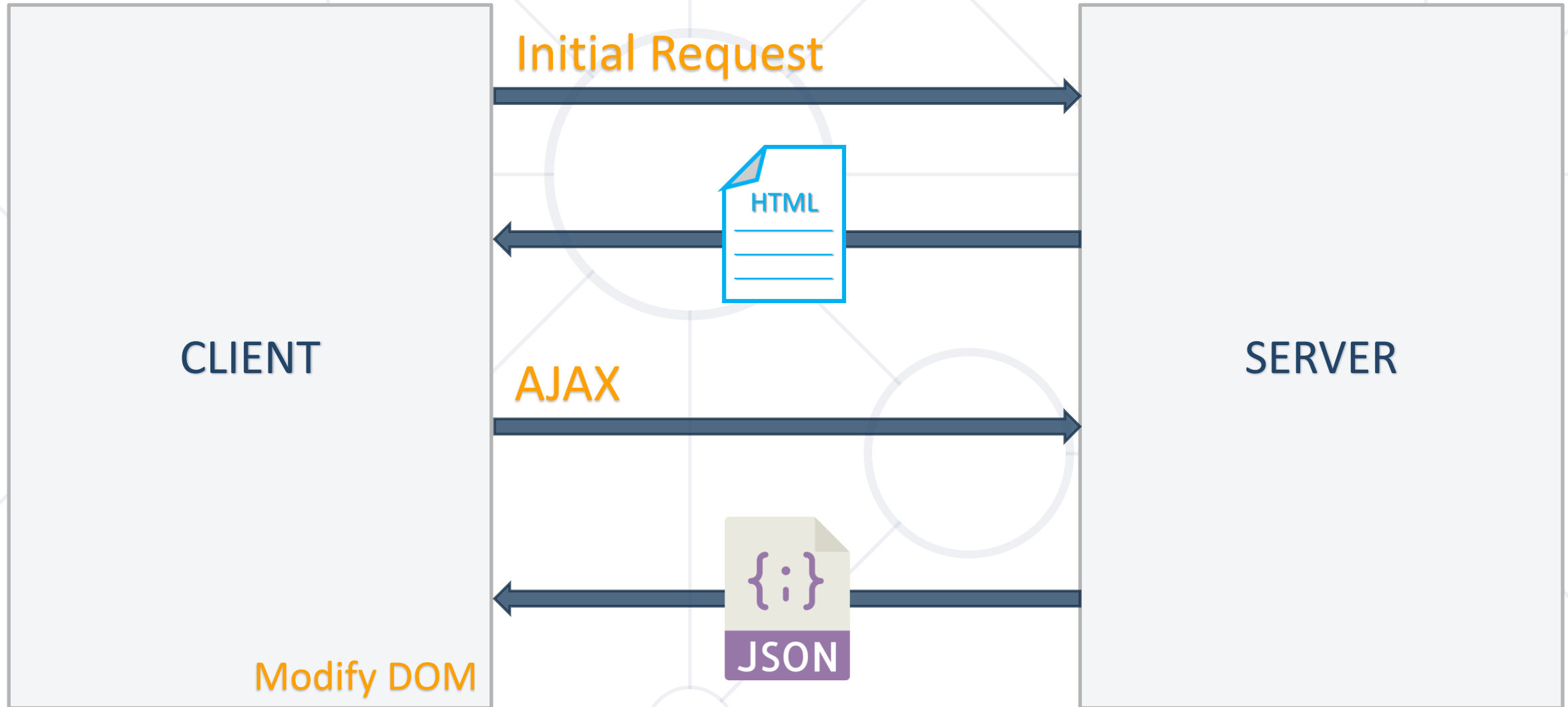


Single Page Applications

- A next evolution from multi-page website
- Web apps that load a **single HTML file**
- SPAs use **AJAX** and **HTML5** to create fluid and responsive Web apps
- **No constant page reloads**



SPA Lifecycle



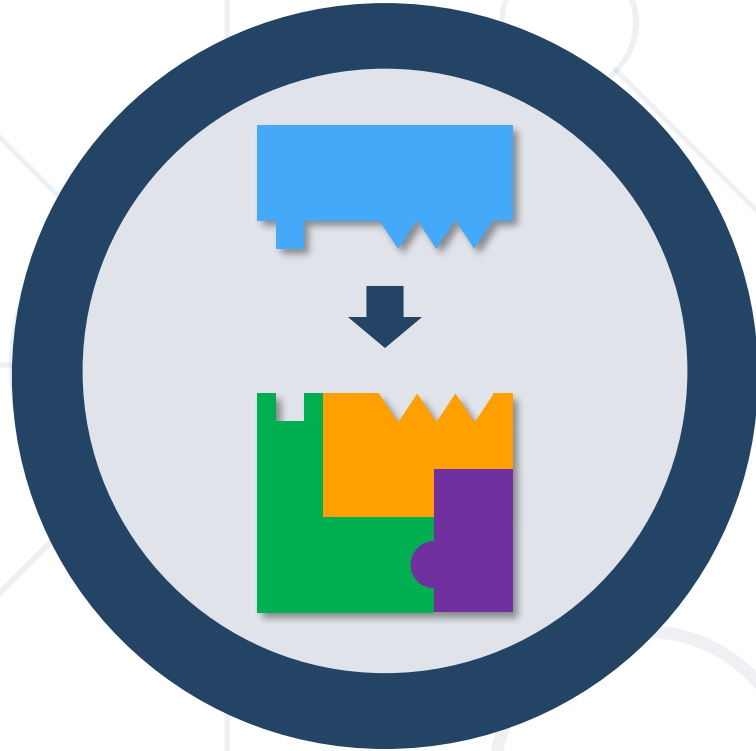
■ Pros

- Load all scripts **only once**
- **Maintain state** across multiple pages
- Browser **history** can be used
- Better **UX**

■ Cons

- Perform poor on the search engine
 - Server-side rendering helps
- Provide **single sharing link**
- Less secure





JavaScript Modules

Definition and Usage

Modules

- 
- A set of **related functionality**
 - Resolve naming collisions
 - Expose only public behavior
 - Not populate the global scope
 - **Loaded** by setting the **type attribute** of a script:

```
<script src="app.js" type="module"></script>
```

- **Note:** Browsers **will not load** modules from the **file system** – you **must** use a local server
 - **lite-server** or similar

- **export** → expose public API

```
export function updateScoreboard(newResult) { ... }  
export const homeTeam = 'Tigers';
```

- You can **export multiple** members

```
export { addResult, homeTeam as host };
```

- **Default exports** can be imported without a name

```
export default function addResult(newResult) { ... }
```

- **import** → load **entire module** (all exports)

```
import * as scoreboard from './scoreboard.js';  
scoreboard.updateScore(); // call from module
```

- Import **specific members** by name


```
import {addResult, homeTeam} from './scoreboard.js';  
addResult(); // call directly by name
```

- Import **default export** by specifying alias

```
import addResult from './scoreboard.js';  
addResult(); // call directly by name
```

Legacy Approach – IIFE Modules

- **IIFE modules** were essential for larger projects
- They hide the unnecessary and expose only needed behavior/objects to the global scope



```
(function(scope) {  
  const selector = 'loading';  
  const loadingElement = document.querySelector(selector);  
  const show = () => loadingElement.style.display = '';  
  const hide = () => ladingElement.style.display = 'none'  
  // Only this is visible to the global scope  
  scope.loading = { show, hide };  
})(window);
```


Module Best Practices



- **Split code** in modules by related functionality
 - Aim for **high cohesion**
- Only export what is **necessary** for consumers
- Prefer **named exports** over defaults
- **Do not** perform operations on export



SPA Approaches

Creating a Single Page Application

SPA Implementation Requirements

- The application has **multiple views**
- All views **share** a common **state**
- **Modular code** is used
- The page is **not reloaded** when changing views
- Content is **loaded via AJAX**
- New **content is created** by JavaScript

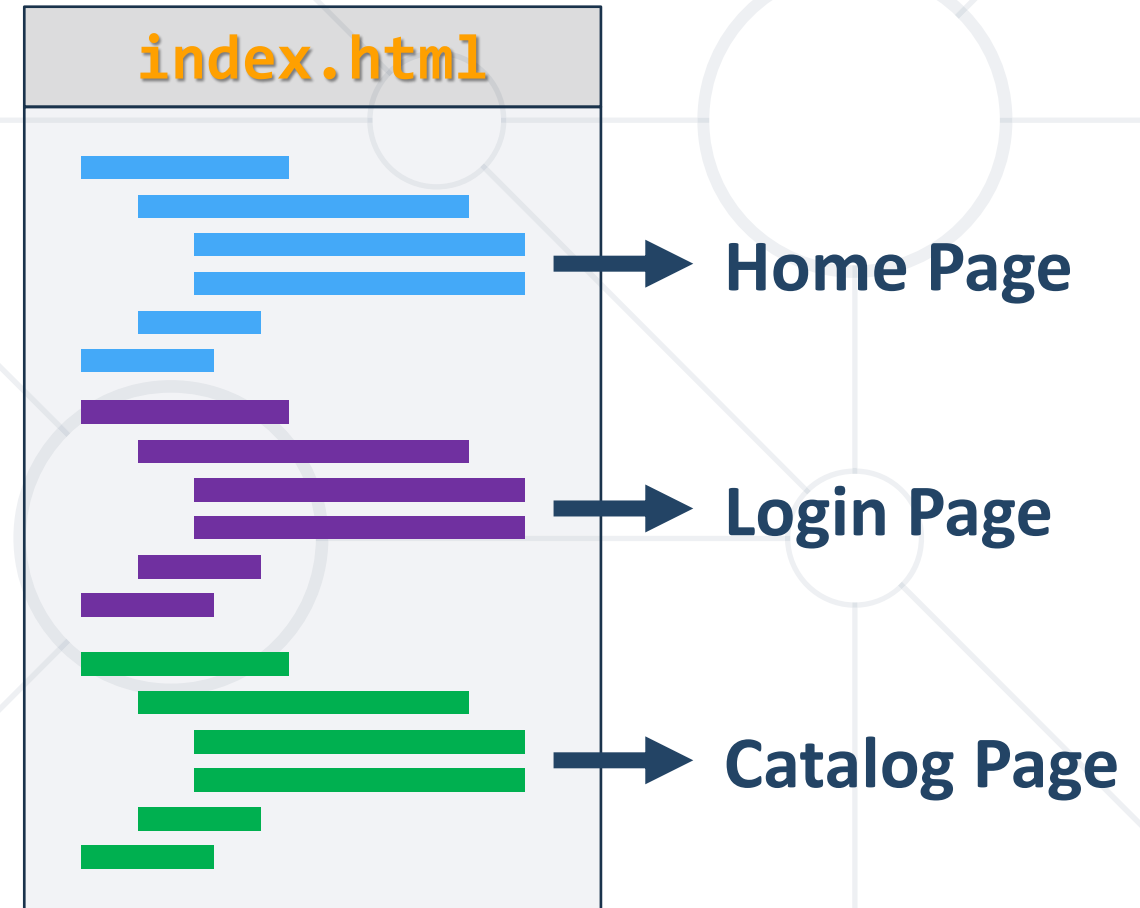


Feasibility Disclaimer



- Contemporary single-page applications employ concepts like **templates** and **routing**
 - Usually with a **front-end framework**
- These topics will be covered in **upcoming lessons**
- The following approach aims to **demonstrate** the **basic principles** and is applicable only for **small apps**
 - Consider it for **educational purposes** only!

- The **HTML template** holds all views as **hidden sections**
- Modules are responsible for **populating** and **displaying** views
- Sections can be controlled by **reference** or by **visibility**
- **Dynamic content** is loaded via AJAX calls



- Each **view** is controlled by its **own module**
 - Contains code for **fetching** and **displaying** related content
- A single script serves as the **application entry point**
 - **Imports** and **initializes** the rest of the modules
 - Holds and manages **shared** (global) **state**

app.js

```
import homePage from './home.js';  
import catalogPage from './catalog.js';  
// Load and initialize all modules
```

- **Anchor tags** instruct the browser to **navigate** to a new page
 - This will **restart** our application
- A **click handler** can be used to prevent this:

```
const navLink = document.getElementById('navLink');
navLink.addEventListener('click', e => {
  e.preventDefault();
  // Load new content and switch the view
});
```

- **View changes** can then be **triggered** from **<a>** elements

Loading and Displaying Content

- Use the **Fetch API** to bring **new content** from the server
- **Modify** or **create** new HTML elements to **display content**

```
async function getArticles() {  
  const response = await fetch(apiUrl);  
  const articles = await response.json();  
  articles.forEach(displayArticle);  
}
```

```
function displayArticle(article) {  
  // Modify DOM tree  
}
```


- Manipulating the DOM tree is a **performance-intensive** process
- When **multiple elements** must be created and populated, place them in a **DocumentFragment**:

```
const fragment = document.createDocumentFragment();  
// Create and populate new elements  
fragment.appendChild(/* element reference */);  
document.body.appendChild(fragment); // Add to body
```



Live Demonstration

Creating a Single Page Application

- A **Web Application** is a **software** accessible using a web browser
 - **Not** the same as a **web site**
- Code can be split and reused with **modules**
- A **Single Page Application** (SPA) operates **without reloading** the page



Questions?



SoftUni Diamond Partners



Coca-Cola HBC
Bulgaria



INFRAGISTICS®



SmartIT



**SOFTWARE
GROUP**

INDEAVR

Serving the high achievers



Postbank

Решения за твоето умре



MOTION SOFTWARE



**SUPER
HOSTING
.BG**

Educational Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, softuni.org

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

