Book Reader — Detailed Technical Documentation

Author: Harshitha Annam
Generated: Detailed developer handbook with formatted code blocks and inbuilt JS method explanations.

Overview

This document provides a thorough, in-depth explanation of the Book Reader JavaScript codebase. It includes: Section-wise breakdown (variables, rendering logic, event handling) Formatted code snippets in monospace Line-by-line explanations for major functions Reference for inbuilt JavaScript APIs used in the code

File & Variable Structure

Top-level DOM references and state variables:

```
let fileInput = document.getElementById('file-input');
let canvas = document.getElementById('file-render');
let context = canvas.getContext('2d');
let rangeInput = document.getElementById('page-range');
var pdfDoc = null;
var pageNumber = 1;
var scale = 1;
var totalPages = 0;
var { pdfjsLib } = globalThis;
var viewport;
let fingerprint;
let noteText = document.getElementById('note-textarea');
let noteBtn = document.getElementById('save-note-btn');
let delBtns = [];
let fileContainer = document.getElementById('file-render-container');
let showNotesBtn = document.getElementById('show-notes-btn');
let threshold =  0.9;
let renderingComplete = false;
let allowObserverUpdates = true;
let pageRendered = false;
if(window.innerWidth<=400)
{
    threshold=0.8;
}
```

Explanation:

- `getElementById(...)` — retrieves a DOM element by its id. Returns `null` if not found. - `canvas.getContext('2d')` — returns a 2D drawing context used by PDF.js to render pages into the canvas pixel buffer. - `pdfDoc` — will be populated with the PDF.js document object after a PDF is loaded via `pdfjsLib.getDocument(...)`. - `pageNumber`, `scale`, `totalPages` — basic pagination and rendering scale state variables. - `globalThis.pdfjsLib` — assumes PDF.js is loaded on the page and attached to `globalThis` (the global object) as `pdfjsLib`. - `fingerprint` — unique id returned by PDF.js for each document; used as a key in `localStorage`. - `threshold` — passed to `IntersectionObserver` to control when an element is considered visible. Lower threshold = easier to trigger on smaller visibility. - `renderingComplete` and `allowObserverUpdates` — control flags to avoid race conditions between rendering and intersection updates.

Event Listeners & UI Controls

Sidebar toggle

```
showNotesBtn?.addEventListener('click', function (){
    document.querySelector('.side-bar').classList.toggle('open');
});
```

Explanation:

- `showNotesBtn?.addEventListener(...)` — optional chaining (`?.`) ensures no error if `showNotesBtn` is `null`. If `showNotesBtn` exists, it attaches a click listener. - `document.querySelector('.side-bar')` — selects the first element matching the CSS selector `.side-bar`. - `.classList.toggle('open')` — toggles the `open` class on the sidebar element, showing/hiding it based on CSS rules.

Intersection Observer

```
const observer = new IntersectionObserver((entries)=>{
    if(!renderingComplete || !allowObserverUpdates) return;
    entries.forEach((entry)=>{
        if(entry.isIntersecting){
            pageNumber = parseInt(entry.target.id);
            getCurrentPage();
            rangeInput.value=pageNumber;
            renderSpecificPage();
            updateLastVisited(fingerprint, entry.target.id);
        }
    })
},{
    root:null,
    threshold:threshold,
})
```

Explanation:

- `IntersectionObserver(callback, options)` monitors visibility of target elements relative to a root (viewport if `root` is `null`). - `entries` array — each `entry` is an `IntersectionObserverEntry` object with properties like `isIntersecting`, `intersectionRatio`, `target`, etc. - `entry.isIntersecting` — boolean, `true` when the target intersects the root according to threshold. - `parseInt(entry.target.id)` — converts the canvas element's `id` (string) to a number to set `pageNumber`. - The observer updates UI state (`rangeInput.value`) and triggers `renderSpecificPage()` and `updateLastVisited()` to persist the current page. - `threshold` controls how much of the target must be visible to be considered intersecting.

Navigation Functions

```
function onPrevPage() {
    if (pageNumber <= 1) {
      return;
    }
    pageNumber--;
    rangeInput.value=pageNumber;
    scrolling(pageNumber);
}

function onNextPage() {
    if (pageNumber >= totalPages) {
      return;
    }
    pageNumber++;
    rangeInput.value = pageNumber;
    scrolling(pageNumber)
}

function renderSpecificPage()
{
    const pageRangeInput = rangeInput.value;
    pageNumber = parseInt(pageRangeInput);
    scrolling(pageNumber);
}
```

Explanation (line-by-line):

- `if (pageNumber <= 1) return;` — prevents underflow of page number. - `pageNumber--` / `pageNumber++` — decrement or increment current page index. - `rangeInput.value = pageNumber` — updates the UI slider to reflect the new page number. - `scrolling(pageNumber)` — utility function that calls `element.scrollIntoView({behavior:'smooth'})` to navigate to the page canvas smoothly. - `parseInt(rangeInput.value)` — ensures string values from input elements are converted to integers before assignment.

Local Storage Management

```
function updateLastVisited(fingerprint, pageNum)
{
    let data = JSON.parse(localStorage.getItem('bookReader')) || [];
    const newDoc = {
                'uId':fingerprint,
                'lastVisited':pageNum,
                'bookmarks':[],
                'notes' : [],
            };
    let existingDoc = data.find(doc => doc.uId === fingerprint);

    if(existingDoc)
    {
        existingDoc.lastVisited = pageNum;
    }
    else{
        data.push(newDoc);
    }
    localStorage.setItem('bookReader', JSON.stringify(data));
}
```

Explanation:

- `localStorage.getItem(key)` — retrieves a string value from browser localStorage, or `null` if missing. - `JSON.parse(...)` — converts JSON string back into an object/array. If `null`, `|| []` falls back to an empty array. - `Array.prototype.find(predicate)` — finds the first element matching predicate (returns `undefined` when not found). - `localStorage.setItem(key, value)` — stores data as strings; `JSON.stringify(data)` converts the object to JSON for storage. - Data model ensures each PDF (`uId`) maps to its own bookmarks, notes, and lastVisited page.

Bookmarks Management

```
function addBookmark()
{
    let data = JSON.parse(localStorage.getItem('bookReader')) || [];
    let existingDoc = data.find(doc => doc.uId === fingerprint);
    if(!existingDoc.bookmarks.includes(pageNumber))
    {
        existingDoc.bookmarks.push(pageNumber);
        document.getElementById(`bookmark`).textContent = '★';
    }
    else{
        const idx = existingDoc.bookmarks.indexOf(pageNumber);
        existingDoc.bookmarks.splice(idx,1);
        document.getElementById(`bookmark`).textContent = '■';
    }
    localStorage.setItem('bookReader', JSON.stringify(data));
    showBookmarks();
}

function checkIfBookmarked(){
    let data = JSON.parse(localStorage.getItem('bookReader')) || [];
    let existingDoc = data.find(doc => doc.uId === fingerprint);
    if(existingDoc.bookmarks.includes(pageNumber))
    {
        document.getElementById(`bookmark`).textContent = '★';
    }
    else{
        document.getElementById(`bookmark`).textContent = '■';
    }
}
```

Explanation:

- `Array.prototype.includes(value)` — checks whether an array contains the specified value (strict equality). - `Array.prototype.indexOf(value)` — returns the first index of the value or -1 if not found. - `Array.prototype.splice(index, 1)` — removes an element at the given index. - `textContent` — writes text into the element, replacing its current contents; safer for plain text (avoids HTML parsing like innerHTML would). - After mutating the bookmarks array, the data is persisted back to localStorage.

Notes System

```
function createNote(text)
{
    let data = JSON.parse(localStorage.getItem('bookReader')) || [];
    let existingDoc = data.find(doc => doc.uId === fingerprint);
    let notes = existingDoc.notes;
    if(text.trim() != '')
    {
        notes.push({ 'page' : pageNumber, 'note':text});
        existingDoc.notes = notes;
        noteText.value = '';
        localStorage.setItem('bookReader', JSON.stringify(data));
        showNotes();
    }
}

function showNotes(){
    let data = JSON.parse(localStorage.getItem('bookReader')) || [];
    let existingDoc = data.find(doc => doc.uId === fingerprint);
    let notes = existingDoc?.notes || [];
    document.getElementById('notes-container').innerHTML='';
    notes?.map((el, index) => {
        if(el.page === pageNumber){
            const noteDiv = document.createElement('div');
            const span = document.createElement('span');
            span.textContent = el.note;
            noteDiv.appendChild(span);
            const btn = document.createElement('button');
            btn.id = index;
            btn.textContent = 'delete';
            noteDiv.appendChild(btn);
            document.getElementById('notes-container').appendChild(noteDiv);
            btn.addEventListener('click',(e)=> deleteNote(e))
        }
    });
}
```

Explanation:

- `String.prototype.trim()` — removes whitespace from both ends; used to avoid saving blank notes. - `Array.prototype.map()` — iterates over the notes array; used here primarily for iteration (not for creating a transformed array). - `document.createElement(tag)` — creates new DOM elements dynamically. - `element.appendChild(child)` — inserts a child node into the DOM; used to build the notes UI. - `addEventListener('click', ...)` registers the delete handler for each dynamically created button. - Optional chaining `existingDoc?.notes` ensures code won't throw if `existingDoc` is `undefined`.

Deleting Notes

```
function deleteNote(e){
    let data = JSON.parse(localStorage.getItem('bookReader')) || [];
    let existingDoc = data.find(doc => doc.uId === fingerprint);
    let notes = existingDoc.notes;
    notes.splice(parseInt(e.target.id), 1);
    existingDoc.notes = notes;
    localStorage.setItem('bookReader', JSON.stringify(data));
    showNotes();
}
```

Explanation:

- `e.target.id` accesses the `id` property of the clicked delete button; `parseInt(...)` converts it to a number. - `splice()` removes the note at that index from the array. - After deletion, localStorage is updated and `showNotes()` refreshes the UI.

PDF Rendering Logic (Using PDF.js)

```
function renderPage(num, canvaPage) {
    if(!pdfDoc) {
        console.log('pdf not loaded yet')
    }
    else{
        pdfDoc.getPage(num).then(function(page) {
          let viewport = page.getViewport({scale: scale});
          canvaPage.height = viewport?.height;
          canvaPage.width = viewport?.width;
          var renderContext = {
            canvasContext: canvaPage.getContext('2d'),
            viewport: viewport
          };
          page.render(renderContext).promise.then(function() {
            // optional callback after render
          });
        });
    }
    getCurrentPage();
    showNotes();
}
```

Explanation:

- `pdfDoc.getPage(num)` returns a Promise resolving to a `PDFPageProxy` object representing the page. - `page.getViewport({ scale })` returns an object with `width` and `height` properties scaled according to `scale`. - `canvaPage.height` / `.width` set the canvas pixel dimensions; important to match the viewport to avoid blurriness. - `page.render(renderContext)` returns a rendering task with a `.promise` property allowing chaining to know when rendering completes. - `canvas.getContext('2d')` provides the drawing context passed to PDF.js renderTask. - After rendering, `getCurrentPage()` and `showNotes()` are called to sync UI and notes for that page.

Rendering All Pages & Observing Visibility

```
function renderPages(totalPages){
    fileContainer.innerHTML = '';
    for(let i =1; i<= totalPages;i++)
    {
        const canvaPage = document.createElement('canvas');
        canvaPage.id = i;
        fileContainer.appendChild(canvaPage);
        renderPage(i, canvaPage);
        observer.observe(canvaPage);
    }
}
```

Explanation:

- `fileContainer.innerHTML = ''` clears any previous canvases and notes; ensure old observers are disconnected if necessary. - Creates one `canvas` per page, assigns id to match page number, appends to container, and triggers rendering. - `observer.observe(canvaPage)` registers the canvas with the `IntersectionObserver` to track visibility changes.

Scrolling Utility

```
function scrolling(id){
    element = document.getElementById(id);
    element.scrollIntoView({behavior:'smooth',block:'nearest', inline:'start'});
}
```

Explanation:

- `document.getElementById(id)` returns the canvas element with that id. - `element.scrollIntoView(options)` scrolls the container so the element is visible. Options: - `behavior: 'smooth'` — smooth scrolling animation. - `block: 'nearest'` — vertical alignment. - `inline: 'start'` — horizontal alignment. - If `element` is `null`, calling `scrollIntoView` would throw; ensure element exists before invoking in production code.

File Input Handler (Loading PDFs)

```
fileInput.addEventListener('change', (event) => {
    observer.disconnect();
    let file = event.target.files[0];
    if(file === undefined)
    {
        document.getElementById('prev').disabled = true;
        document.getElementById('next').disabled = true;
        pdfDoc = null;
        context.clearRect(0,0,canvas.width, canvas.height);
        document.getElementById('page_count').textContent = 0;
        document.getElementById('page_num').textContent = 0;
        document.getElementById('bookmark').disabled=true;
    }
    if(file && file.type === 'application/pdf')
    {
        const fileURL = URL.createObjectURL(file);
        pdfjsLib.getDocument(fileURL).promise.then(function(pdf) {
            pdfDoc = pdf;
            document.getElementById('prev').disabled = false;
            document.getElementById('next').disabled = false;
            document.getElementById('bookmark').disabled=false;
            fingerprint = pdf.fingerprint;
            pageNumber = getLastVisitedPage(fingerprint);
```

```
                rangeInput.setAttribute('max', pdf.numPages);
                rangeInput.value = pageNumber;
                totalPages = pdf.numPages;
                document.getElementById('page_count').textContent=totalPages;
                renderPages(totalPages);
                renderingComplete = true;
                setTimeout(()=>{
                    scrolling(pageNumber);
                    allowObserverUpdates = true;
                },200)
                showBookmarks();
          });
        }
    });
```

Explanation:

- `observer.disconnect()` stops the observer from firing while the file changes are being applied to avoid inconsistent callbacks. - `event.target.files[0]` returns the first selected File object (File API). - `URL.createObjectURL(file)` creates a blob URL usable as `pdfjsLib` input without uploading to a server. - `pdfjsLib.getDocument(fileURL).promise` returns a Promise resolving to a `PDFDocumentProxy` object with `.numPages` and `.fingerprint`. - UI elements are enabled/disabled based on whether a valid PDF is loaded. - `setTimeout` with small delay ensures canvases are present and renderingComplete when calling `scrolling(pageNumber)`.

Inbuilt JavaScript Methods & Browser APIs Used (Reference)

document.getElementById(id) — Returns the element with the specified ID. Time complexity O(1) for direct lookup. Returns `null` if not present.
document.querySelector(selector) — Returns first element matching the CSS selector. If none found, returns null.
element.addEventListener(type, listener) — Attaches an event handler. Multiple listeners supported.
Array.prototype.find(predicate) — Returns the first element matching predicate or undefined.
Array.prototype.includes(value) — Returns boolean whether value exists in array.
Array.prototype.splice(index, count) — Removes or replaces elements at index and returns removed elements.
Array.prototype.indexOf(value) — Returns the first index of value or -1.
JSON.parse(string) — Parses JSON string into object; throws on invalid JSON.
JSON.stringify(obj) — Converts object to JSON string for storage or network transport.
localStorage.getItem(key) — Retrieves string from localStorage or null.
localStorage.setItem(key, value) — Stores string value under key.
URL.createObjectURL(file) — Creates a temporary blob URL for a File/Blob object.
element.scrollIntoView(options) — Scrolls the element into the visible area of the browser window.
Promise.then(onFulfilled) — Attaches a callback for the resolved value of a Promise.
Optional Chaining (obj?.prop) — Short-circuits when preceding expression is null/undefined.
Template literals `...${expr}...` — Embed expressions in strings and create dynamic strings.
document.createElement(tag) — Creates an HTML element of given tag.
element.appendChild(child) — Appends a child DOM node.
element.textContent — Sets or gets text content of the element; safer than innerHTML for plain text.

Best Practices & Performance Tips

- Disconnect and reconnect observers when making bulk DOM changes to avoid expensive callbacks. - Batch localStorage writes where possible; prefer writing once after multiple mutations. - Throttle/debounce scroll and heavy UI events when adding more interactivity. - Resize canvases to devicePixelRatio for sharper rendering on high-DPI displays: - e.g., `canvas.width = viewport.width * devicePixelRatio; canvas.style.width = viewport.width + 'px';` - Clean up event listeners on removed DOM nodes to avoid memory leaks. - Consider virtualizing pages: render only nearby pages and lazy-load distant pages to conserve memory.

Future Enhancements

- Add text search inside PDFs using PDF.js text layer extraction. - Support annotations (highlight, underline) and save them in localStorage or backend. - Add user authentication and cloud sync for notes/bookmarks. - Add robust error handling and user messages for corrupted PDFs. - Add keyboard navigation (arrow keys, PgUp/PgDn) and accessibility improvements (ARIA labels).

Full Code Listing (Original)

```
<Full code excerpt omitted to keep PDF concise — include full listing on request>
```