Bookstack

Set public role for specific content

Table of Contents

Overview

By default, the Public role often has "View All" permissions enabled. When you turned on "Allow public viewing," those role-level permissions immediately applied to every book and page on your site. To fix this and keep only specific pages public (the "whitelist" approach), follow these steps:

1. Strip the Public Role of its "View All" Power This is the most important step to prevent everything from being visible.

Steps

To allow everyone (unauthenticated guests) to view a specific page in BookStack, you must enable Public Viewing in the system settings and then grant the Public role permission to view that specific content.

Step 1: Enable Public Access

Before you can make a single page public, the entire instance must allow public visitors.

1. Log in as an Admin.
2. Navigate to Settings > Application Settings.
3. Locate the "Security" section and check the box for "Allow public viewing?".
4. Save the settings.

Step 2: Configure the Public Role

By default, the Public role might have access to more than you intend. It is recommended to restrict its base permissions so you can "whitelist" only specific pages.

1. Go to Settings > Roles.
2. Edit the Public role (sometimes called "Guest").
3. Uncheck all "Asset Permissions" (like View All/Own, Create, etc.) if you want a fully private site with only specific public pages.
4. Save the role.

Step 3: Set Permissions for the Specific Page

Now, grant the Public role access to the individual item.

1. Navigate to the Page (or Book/Chapter) you want to share.
2. Click the Permissions action in the sidebar or dropdown.
3. Check the box for "Enable Custom Permissions".
4. Under the "Role Overrides" section, find the Public role and check the View permission.
5. Click Save Permissions.

> **ℹ Note**
> If you set these permissions at the Book level, they will automatically cascade down to all chapters and pages within that book unless they have their own custom permissions set.

If all pages are public now

This happens because, by default, the Public role often has "View All" permissions enabled. When you turned on "Allow public viewing," those role-level permissions immediately applied to every book and page on your site.

1. Strip the Public Role of its "View All" Power

- Go to `Settings > Roles`.
- Edit the Public role (sometimes called "Guest").
- Scroll down to Asset Permissions.
- Uncheck every box under "View", "Create", "Edit", and "Delete" for Books, Chapters, and Pages.
- Save the role.
- Result: Now, even though public viewing is "On," guests can't see anything because their role has no permission to view any assets.

2. Grant Permission to ONLY the Specific Content

Now that the door is locked for everything else, you can unlock just the items you want.

- Navigate to the specific Book or Page you want to make public.
- Click Permissions in the sidebar.
- Check "Enable Custom Permissions".
- In the Role Overrides section, find the Public role and check only the View box.
- Save the permissions.

> **Important**
> Watch out for "Everyone Else" In the Page/Book permission settings, there is often an "Everyone Else" row at the bottom.

- Make sure "Inherit defaults" is unchecked for that row if you want to be extra sure nothing leaks.
- If it is checked, it will fall back to whatever you set in the main Public Role settings (which we just cleared in Step 1).

> **Tip**
> Permissions in BookStack cascade downwards (Book -> Chapter -> Page). If you make a Book public, every page inside it becomes public too. If you only want one page public, set the permissions only on that page.

Customisation via Javascript / CSS

[TOC]

Overview

Code

The following code enables the support for **rendering** additional features, like:

- Highlight lines in code block
- Admonitions / Callouts Github syntax
- Make the top navigation bar stick to the top
- Mermaid
- Table Of Content

```
<!-- Support for highlight lines in code block -->
<style>
  .cm-line-highlight {
    background-color: #fffbcc !important;
    border-left: 3px solid #f0c000;
    padding-left: calc(1em - 3px) !important;
  }
</style>
<script>
(function () {
  const MARKERS = ['', '', ''];


  function highlightLines(root) {
    (root || document).querySelectorAll('.cm-line').forEach(function (line) {
      const text = line.textContent;
      if (!MARKERS.some(m => text.includes(m))) return;

      line.style.backgroundColor = '#fffbcc';
      line.style.borderLeft = '3px solid #f0c000';
      line.style.paddingLeft = 'calc(1em - 3px)';
      line.style.display = 'block';

      var allSpans = Array.from(line.querySelectorAll('span'));
      var markerChars = '#[hl]';
      var toHide = [];
      var consumed = '';
```

```
      for (var i = allSpans.length - 1; i >= 0; i--) {
        var span = allSpans[i];
        if (span.querySelector('span')) continue;
        consumed = span.textContent + consumed;
        toHide.push(span);
        if (consumed.replace(/\s+/g, '') === markerChars) break;
        if (consumed.replace(/\s+/g, '').length > markerChars.length) break;
      }

      if (consumed.replace(/\s+/g, '') === markerChars) {
        toHide.forEach(function (span) {
          span.style.fontSize = '0';
          span.style.letterSpacing = '-1em';
          span.style.opacity = '0';
          span.style.userSelect = 'none';
          span.style.pointerEvents = 'none';
          span.style.display = 'inline-block';
          span.style.overflow = 'hidden';
          span.style.width = '0';
        });
      }
    });
  }

  function debounce(fn, delay) {
    var timer;
    return function () {
      clearTimeout(timer);
      timer = setTimeout(fn, delay);
    };
  }

  function attachToEditor(editor) {
    var content = editor.querySelector('.cm-content');
    if (!content || content._hlObserver) return;

    // Debounced re-apply — waits for CodeMirror to finish its own re-render
    var reapply = debounce(function () {
      highlightLines(editor);
    }, 50);

    var obs = new MutationObserver(reapply);
    obs.observe(content, { childList: true, subtree: true, attributes: true, characterData: true
});
    content._hlObserver = obs;

    // Also re-apply on any click/focus inside the editor since CM re-renders on interaction
    editor.addEventListener('click', function () {
      setTimeout(function () { highlightLines(editor); }, 0);
      setTimeout(function () { highlightLines(editor); }, 50);
      setTimeout(function () { highlightLines(editor); }, 150);
    });
    editor.addEventListener('focusin', function () {
      setTimeout(function () { highlightLines(editor); }, 0);
```

```
      setTimeout(function () { highlightLines(editor); }, 50);
      setTimeout(function () { highlightLines(editor); }, 150);
    });

    highlightLines(editor);
  }

  document.addEventListener('DOMContentLoaded', function () {
    document.querySelectorAll('.cm-editor').forEach(attachToEditor);

    var bodyObserver = new MutationObserver(function (mutations) {
      mutations.forEach(function (mutation) {
        mutation.addedNodes.forEach(function (node) {
          if (node.nodeType !== 1) return;
          if (node.classList && node.classList.contains('cm-editor')) attachToEditor(node);
          node.querySelectorAll && node.querySelectorAll('.cm-editor').forEach(attachToEditor);
        });
      });
    });
    bodyObserver.observe(document.body, { childList: true, subtree: true });

    [200, 500, 1000, 2000].forEach(function (ms) {
      setTimeout(function () {
        document.querySelectorAll('.cm-editor').forEach(attachToEditor);
      }, ms);
    });
  });
})();
</script>

<!-- Admonitions / Callouts Github syntax -->
<style>
  .callout { border-radius: 6px; padding: 12px 16px; margin: 16px 0; border-left: 4px solid; }
  .callout-title { font-weight: 600; margin-bottom: 4px; display: flex; align-items: center; gap:
6px; }
  .callout-note     { background: #dbeafe; border-color: #3b82f6; color: #1e40af; }
  .callout-tip      { background: #dcfce7; border-color: #22c55e; color: #15803d; }
  .callout-important{ background: #ede9fe; border-color: #a855f7; color: #6b21a8; }
  .callout-warning  { background: #fef9c3; border-color: #eab308; color: #854d0e; }
  .callout-caution  { background: #fee2e2; border-color: #ef4444; color: #991b1b; }
  .callout p { margin: 0; }

  /* Hide any icon BookStack injects into or around blockquotes */
  .callout svg,
  .callout img,
  .callout i,
  .callout span.icon,
  .callout::before,
  .callout::after { display: none !important; }
</style>
<script>
document.addEventListener('DOMContentLoaded', function () {
  const TYPES = {
    NOTE:      { icon: '𝐢', cls: 'callout-note',      label: 'Note' },
```

```
    TIP:       { icon: '□', cls: 'callout-tip',       label: 'Tip' },
    IMPORTANT: { icon: '□', cls: 'callout-important', label: 'Important' },
    WARNING:   { icon: '⚠', cls: 'callout-warning',   label: 'Warning' },
    CAUTION:   { icon: '□', cls: 'callout-caution',   label: 'Caution' },
  };

  document.querySelectorAll('.page-content blockquote').forEach(function (bq) {
    const first = bq.querySelector('p');
    if (!first) return;

    const match = first.textContent.trim().match(/^\[!(NOTE|TIP|IMPORTANT|WARNING|CAUTION)\]/i);
    if (!match) return;

    const type = TYPES[match[1].toUpperCase()];

    // Remove the [!TYPE] line
    const remaining = first.innerHTML.replace(/^\[!(NOTE|TIP|IMPORTANT|WARNING|CAUTION)\]
(<br\s*\/?>)?/i, '').trim();
    if (remaining) first.innerHTML = remaining;
    else first.remove();

    // Build callout div
    const div = document.createElement('div');
    div.className = `callout ${type.cls}`;
    div.innerHTML = `<div class="callout-title">${type.icon} ${type.label}</div>`;
    Array.from(bq.childNodes).forEach(n => div.appendChild(n.cloneNode(true)));

    bq.replaceWith(div);
  });
});
</script>

<!-- Make the top navigation bar stick to the top -->
<style>
  header {
    position: fixed;
    top: 0;
    width: 100%;
    z-index: 1000;
  }

  /* Add padding to the top of the body to prevent content from hiding behind the header */
  body {
    padding-top: 60px; /* Adjust this value based on your header height */
  }
</style>

<!-- Enable Mermaid -->
<script src="https://cdn.jsdelivr.net/npm/mermaid/dist/mermaid.min.js"></script>
<style>
  .mermaid {
    border-radius: 6px;
    padding: 16px;
  }
```

```
</style>
<script>
  document.addEventListener('DOMContentLoaded', function () {
    document.querySelectorAll('code.language-mermaid').forEach(function(el) {
      const div = document.createElement('div');
      div.className = 'mermaid';
      div.textContent = el.textContent;
      el.parentNode.replaceWith(div);
    });

    mermaid.initialize({
      startOnLoad: true,
      theme: 'dark',
    });
  });
</script>

<!-- Display TOC -->
<script>
  document.addEventListener('DOMContentLoaded', function () {
    const content = document.querySelector('.page-content');
    if (!content) return;

    // Find [TOC] placeholder (in a paragraph or standalone)
    const walker = document.createTreeWalker(content, NodeFilter.SHOW_TEXT);
    let tocNode = null;
    while (walker.nextNode()) {
      if (walker.currentNode.textContent.trim() === '[TOC]') {
        tocNode = walker.currentNode;
        break;
      }
    }
    if (!tocNode) return;

    // Gather headings
    const headings = Array.from(content.querySelectorAll('h1, h2, h3, h4'));
    if (!headings.length) return;

    // Ensure headings have IDs
    headings.forEach((h, i) => {
      if (!h.id) h.id = 'toc-heading-' + i;
    });

    // Build ToC
    const nav = document.createElement('nav');
    nav.className = 'inline-toc';
    nav.innerHTML = '<strong>Table of Contents</strong>';
    const ul = document.createElement('ul');

    const minLevel = Math.min(...headings.map(h => parseInt(h.tagName[1])));

    headings.forEach(h => {
      const level = parseInt(h.tagName[1]) - minLevel;
      const li = document.createElement('li');
```

```
      li.style.marginLeft = (level * 16) + 'px';
      const a = document.createElement('a');
      a.href = '#' + h.id;
      a.textContent = h.textContent;
      li.appendChild(a);
      ul.appendChild(li);
    });

    nav.appendChild(ul);

    // Style it
    const style = document.createElement('style');
    style.textContent = `
      .inline-toc {
        display: inline-block;
        background: var(--color-bg, #f8f9fa);
        border: 1px solid var(--color-border, #dee2e6);
        border-radius: 6px;
        padding: 12px 20px;
        margin: 16px 0;
        font-size: 0.9em;
        min-width: 200px;
        max-width: 400px;
      }
      .inline-toc strong {
        display: block;
        margin-bottom: 8px;
        font-size: 1em;
      }
      .inline-toc ul {
        list-style: none;
        margin: 0;
        padding: 0;
      }
      .inline-toc li {
        margin: 4px 0;
      }
      .inline-toc a {
        color: var(--color-link, #1a73e8);
        text-decoration: none;
      }
      .inline-toc a:hover {
        text-decoration: underline;
      }
    `;
    document.head.appendChild(style);

    // Replace [TOC] node with the nav
    tocNode.parentNode.replaceWith(nav);
  });
</script>
```

Export the content page as it is

[TOC]

Overview

Customizations aren't appearing in PDF exports becuase BookStack's PDF engine (by default, dompdf) is a separate process that doesn't execute JavaScript and only recognizes a subset of CSS.

To ensure your styles are applied to PDF exports follows the steps

Steps

1. Install npm

```
apt update
apt install npm -y
```

2. Install Puppeteer

```
npm install puppeteer
```

Install Chrome for the Web User

```
# Define a local cache path and install
PUPPETEER_CACHE_DIR=/opt/bookstack/.cache npx puppeteer browsers install chrome
```

Set Cache Permissions

```
# Replace 'www-data' with your web user if different (e.g., 'nginx')
chown -R www-data:www-data /opt/bookstack/.cache
```

Update .env to manipulated export command

Edit `.env` adding:

```
ALLOW_UNTRUSTED_SERVER_FETCHING=true
EXPORT_PDF_COMMAND="node /opt/bookstack/puppeteer-pdf.js {input_html_path} {output_pdf_path}"
EXPORT_PDF_COMMAND_TIMEOUT=30
```

Create the Export Script

In BookStack root directory (`/opt/bookstack`) create a new file named `puppeteer-pdf.js`

```
const puppeteer = require('puppeteer');
const fs = require('fs');

(async () => {
    const browser = await puppeteer.launch({
        executablePath: '/opt/bookstack/.cache/chrome/linux-146.0.7680.76/chrome-linux64/chrome',
        args: ['--no-sandbox', '--disable-setuid-sandbox']
    });
    const page = await browser.newPage();

    const htmlContent = fs.readFileSync(process.argv[2], 'utf8');
    await page.setContent(htmlContent, { waitUntil: 'networkidle0' });

    // --- NEW TOC GENERATION LOGIC ---
    await page.evaluate(() => {
        const tocPlaceholder = document.body.innerHTML.match(/\[TOC\]/g);
        if (!tocPlaceholder) return;

        // Find all headings (h1 to h4)
        const headings = Array.from(document.querySelectorAll('h1, h2, h3, h4'));
        let tocHtml = '<div class="table-of-contents"><h2>Table of Contents</h2><ul>';

        headings.forEach((heading, index) => {
            const id = heading.id || `heading-${index}`;
            heading.id = id; // Ensure heading has an ID to link to
            const level = heading.tagName.toLowerCase();
            tocHtml += `<li class="toc-${level}"><a href="#${id}">${heading.innerText}</a></li>`;
        });

        tocHtml += '</ul></div>';

        // Replace the [TOC] text with our generated HTML
        document.body.innerHTML = document.body.innerHTML.replace('[TOC]', tocHtml);
    });
    // -------------------------------
```

```
    await page.pdf({
        path: process.argv[3],
        format: 'A4',
        printBackground: true,
        margin: { top: '1cm', right: '1cm', bottom: '1cm', left: '1cm' }
    });

    await browser.close();
})();
```