ChatGPT

Diagnosis of the Issue

The root problem is that the GrapesJS-generated subproduct HTML/CSS is not inherently responsive, so inserting it raw into the React app breaks the layout on mobile. By default, GrapesJS outputs fixed-width layouts (often using pixel values) and does not add a viewport meta tag, so mobile browsers render the page at desktop width. As a result, elements overflow or shrink incorrectly, and site-wide elements (navbars/footers) may be pushed out of view or hidden by the inserted content. In summary:

- **Fixed widths and no mobile meta:** GrapesJS content often uses fixed pixel widths and lacks a `<meta name="viewport" …>` declaration, so smartphones treat it like a large desktop page. This prevents natural scaling.
- **No automatic media queries:** Unless explicitly configured, GrapesJS won't generate CSS media queries for smaller screens. Without those breakpoints, layouts won't adapt to narrow viewports.
- **CSS conflicts and container issues:** Injected styles (or inline `<style>` tags) may not be scoped to the subproduct page. Unscoped rules or fixed dimensions can override the site's responsive CSS, hiding headers/footers or breaking the global grid.

These combine to break the mobile layout: the GrapesJS content neither scales (due to missing viewport and fixed widths) nor wraps (due to missing flex/grid or media rules), causing overflow and hidden site elements.

React Frontend Updates for Responsiveness

To contain GrapesJS content and make it responsive, wrap the injected HTML in a dedicated container with scoped CSS. For example, update the subproduct detail component as follows:

```
function SubproductDetailPage({ htmlContent /*, cssContent if any */ }) {
  return (
    <div className="subproduct-page">
      {/* If GrapesJS CSS is stored separately, you could inject it like this:
          <style>{cssContent}</style>
          Or ensure it's imported globally. */}
      <div
        className="subproduct-content"
        dangerouslySetInnerHTML={{ __html: htmlContent }}
      />
    </div>
  );
}
```

Then add CSS rules to constrain widths and scale media within this container. For example (in your stylesheet or as a CSS-in-JS block):

```
.subproduct-page {
  width: 100%;
  box-sizing: border-box;
  /* Ensure the content takes full container width */
}
.subproduct-content {
  width: 100%;
  box-sizing: border-box;
}
.subproduct-content img,
.subproduct-content video,
.subproduct-content iframe {
  max-width: 100%;
  height: auto;
}
/* Optional: ensure any fixed-width elements shrink if needed */
.subproduct-content * {
  max-width: 100%;
  box-sizing: border-box;
}
```

These changes ensure that inserted elements (images, videos, etc.) cannot overflow their container and instead scale to 100% width on small screens. The `.subproduct-*` classes scope these rules only to the GrapesJS content, so other pages remain unaffected.

Additionally, verify that the main HTML `<head>` includes a viewport meta tag (usually in `public/index.html` for React apps):

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

This ensures the browser uses the device's actual width for layout. If the rest of the site is already responsive, this is likely already present.

Together, these React-side updates (container classes and CSS rules) wrap and constrain the GrapesJS content, allowing it to flow with the rest of the page without breaking global layout or hiding navbars/footers.

GrapesJS Editor Configuration for Mobile-Friendly Output

The key to automatically generating responsive GrapesJS content is to configure its **Device Manager** and **media queries**. In the GrapesJS init code, specify multiple devices and breakpoints. For example, using a mobile-first approach (so mobile styles are default and desktop is in a min-width query) you can do:

```
const editor = grapesjs.init({
  // ... other config ...
  mediaCondition: 'min-width', // Use min-width media queries (mobile-first)
  deviceManager: {
    devices: [
      { name: 'Mobile',  width: '320px', widthMedia: ''    },  // default/mobile
      { name: 'Desktop', width: '',       widthMedia: '1024px' }
    ],
  },
  panels: {
    defaults: [
      // (Optional) add device-switch buttons to the UI
      {
        id: 'panel-devices',
        el: '.panel__devices',
        buttons: [
          { id: 'device-mobile',  label: '📱', command: 'set-device-mobile' },
          { id: 'device-desktop', label: '💻', command: 'set-device-desktop',
active: true },
        ],
      },
    ],
  },
  commands: {
    defaults: [
      {
        id: 'set-device-mobile',
        run: editor => editor.setDevice('Mobile')
      },
      {
        id: 'set-device-desktop',
        run: editor => editor.setDevice('Desktop')
      },
    ]
  }
});
```

```
// Set initial device to Mobile for a mobile-first start:
editor.setDevice('Mobile');
```

This configuration (as documented by GrapesJS) creates two breakpoints: **Mobile (<=320px)** and **Desktop (>1024px)**. With `mediaCondition: 'min-width'`, GrapesJS will export all default styles as "mobile" and wrap the desktop-specific overrides in `@media (min-width:1024px) { … }`. In contrast, the default behavior (without `mediaCondition`) uses `max-width` queries (placing smaller styles in max-width rules). Both approaches can work, but mobile-first (`min-width`) often makes it easier to design for small screens first.

For example, the GrapesJS docs show defining a mobile device like this to automatically generate a CSS media query at 480px:

```
deviceManager: {
  devices: [
    { name: 'Desktop', width: '' },
    { name: 'Mobile',  width: '320px', widthMedia: '480px' },
  ],
},
```

This causes GrapesJS to output styles inside `@media (max-width: 480px)` for mobile-specific rules [1]. In our mobile-first setup, we instead define the desktop breakpoint (1024px) in a `min-width` media query (no widthMedia on Mobile). The GrapesJS docs illustrate a mobile-first example like this [2].

Also ensure that in GrapesJS's Style Manager, components like **Width** and **Max-width** allow percentage units (they do by default, e.g. `units: ['px','%']`) so content editors can set fluid widths rather than fixed pixels. Encourage using percentage (%) widths or flexbox/grid properties within GrapesJS to create fluid layouts that shrink naturally on small screens.

In summary: Configure GrapesJS's `deviceManager` with at least two devices (Mobile and Desktop) and use `mediaCondition: 'min-width'` for mobile-first CSS. This ensures the exported HTML/CSS includes responsive breakpoints automatically. (For reference, see GrapesJS docs on device settings [1] [2].)

Ensuring Future Content is Responsive

To make sure all future admin-created subproduct pages are responsive by default:

- **Design in Mobile View:** Instruct content editors to use the GrapesJS **device switcher** (as configured above) to preview and tweak layouts in the Mobile view. They should arrange blocks to stack or shrink appropriately at the mobile breakpoint.
- **Use Flexible Units:** Encourage use of percentages, `flex`, or `grid` in GrapesJS blocks instead of fixed pixel widths. For example, setting a block's width to `100%` or using a flex container will naturally fill smaller screens.

- **Responsive Components:** Add (or customize) GrapesJS blocks that are inherently responsive. For instance, a two-column block that collapses to one column under 768px. You can predefine such blocks using the style manager's `styleMedia` field (as in advanced block definitions) to include media rules.
- **Preview Regularly:** Advise admins to frequently switch between devices in the editor (using the panel buttons) to see how content adapts. With the deviceManager set, GrapesJS will automatically apply the right CSS rules in each view.
- **Limit Global Styles:** Keep GrapesJS-generated styles scoped to the subproduct content. Avoid adding global CSS in GrapesJS that might affect site navigation. If needed, wrap components in a container with a unique class.

By combining the device breakpoints above with a mobile-first workflow (designing for small screens first, then adding styles for larger screens), new pages will automatically include the necessary CSS media queries. The exported HTML/CSS will then scale on mobile without further tweaks.

Applying Changes to Existing Content

For already-uploaded subproduct pages (HTML/CSS stored in the database), follow these steps:

1. **Re-import and Save with New Config:** Ideally, load each existing page's JSON or HTML into the updated GrapesJS editor (with the new `deviceManager` settings) and re-save. This regenerates the HTML/CSS with responsive breakpoints. If the original JSON is not available, you can paste the old HTML into the GrapesJS canvas, adjust as needed in both Desktop and Mobile views, and export again.
2. **Add Container Styling:** For pages you cannot easily re-edit, apply the same React/CSS wrapper solution as above (`.subproduct-content` with `max-width:100%`, etc.). This will mitigate many issues without altering the content itself.
3. **Global CSS Overrides:** As a fallback, introduce site-wide media queries to patch common fixed-width issues. For example, you could add a rule like:

   ```
   @media (max-width: 767px) {
     .subproduct-content img,
     .subproduct-content div {
       width: 100% !important;
     }
   }
   ```

   This forces images/blocks to fill smaller screens. However, this is a last resort and may have edge cases.
4. **Review and Test:** After applying these fixes, test each existing page on real mobile devices or emulators. Check that navbars and footers appear and that content wraps or scrolls gracefully.

While retrofitting has limits, using the new GrapesJS configuration and the scoped CSS wrapper will resolve most responsiveness issues. Any remaining formatting quirks can be addressed case-by-case with minor CSS adjustments.

Conclusion

The lack of mobile responsiveness was caused by GrapesJS output using fixed dimensions and no mobile meta or breakpoints. The solution is two-fold:

- **React-side:** Wrap the GrapesJS HTML in a full-width container and apply scoped CSS (e.g. `max-width: 100%` on images/elements) so the content scales to the device. Ensure the viewport meta tag is present.
- **GrapesJS-side:** Configure the editor's Device Manager to use a mobile-first setup. Define a Mobile device (e.g. 320px width) and a Desktop device, and use `mediaCondition: 'min-width'` so that mobile styles are default and desktop styles go in `@media` rules [2]. This makes all future exports automatically include responsive CSS.

These targeted changes are safe for the rest of the site. The React wrapper styles only apply to subproduct pages, and the GrapesJS config changes affect only the editor output. No existing site components or global styles are altered. As a result, all new and updated subproduct pages will render properly on mobile without further manual CSS tweaks.

Sources: GrapesJS documentation on responsive (deviceManager) configuration [1] [2].

---

[1] [2] Getting Started | GrapesJS
https://grapesjs.com/docs/getting-started.html