Improving Rslidy: Responsive Web-Based Slide Decks

Fabian Platzer

Improving Rslidy: Responsive Web-Based Slide Decks

Fabian Platzer B.Sc.

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's Degree Programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Human-Centred Computing (HCC)

Graz, 30 Jun 2026

Verbesserung von Rslidy: Responsive, webbasierte Foliensätze

Fabian Platzer B.Sc.

Masterarbeit

für den akademischen Grad

Diplom-Ingenieur

Masterstudium: Software Entwicklung Wirtschaft

an der

Technischen Universität Graz

Begutachter

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Human-Centred Computing (HCC)

Graz, 30 Jun 2026

Diese Arbeit ist in englischer Sprache verfasst.

Abstract

Web-based slide decks have become a popular alternative to traditional presentation software. Modern browsers support a wide range of technologies that allow presentations to be created and displayed entirely using open web standards such as HTML, CSS, and JavaScript. This Master's thesis examines how these technologies can be used to implement responsive, flexible, and maintainable presentation systems.

The focus lies on the design and development of Rslidy v2, an extension and modernisation of the Rslidy slide deck framework originally developed at Graz University of Technology. The technological foundations of web-based slide decks are first examined. These slide decks are categorised into text-based slide decks, JavaScript-based slide decks, hosted slide decks, and responsive slide decks. This classification highlights different design approaches and their trade-offs with respect to responsiveness, usability, and extensibility.

Building on this analysis, several improvements introduced in Rslidy v2 are presented. The development environment was modernised through a revised build process and improved dependency management. A redesigned print configuration system allows slide layout, scaling, and slide selection to be controlled directly within the presentation interface through dynamically generated CSS rules. In addition, responsive tables with automatic data labels and interactive sorting improve the presentation of structured data across different screen sizes.

Together, these contributions demonstrate how modern web technologies can be applied to create lightweight yet powerful slide decks. The resulting framework improves responsiveness, print consistency, and maintainability while preserving the browser-native approach of the original Rslidy system.

Kurzfassung

Webbasierte Foliensätze sind zu einer beliebten Alternative zu klassischer Präsentationssoftware geworden. Moderne Webbrowser unterstützen eine Vielzahl von Technologien, die es ermöglichen, Präsentationen vollständig auf Basis offener Webstandards wie HTML, CSS und JavaScript zu erstellen und darzustellen. Diese Masterarbeit untersucht, wie diese Technologien eingesetzt werden können, um responsive, flexible und wartbare Foliensätze zu implementieren.

Der Schwerpunkt liegt auf dem Entwurf und der Entwicklung von Rslidy v2, einer Erweiterung und Modernisierung des Rslidy Slide Deck Frameworks, das ursprünglich an der Technischen Universität Graz entwickelt wurde. Zunächst werden die technologischen Grundlagen von webbasierten Foliensätze untersucht. Diese Foliensätze werden in textbasierte Foliensätze, JavaScript-basierte Foliensätze, Hosted Foliensätze und responsive Foliensätze kategorisiert. Diese Klassifikation zeigt unterschiedliche Designansätze sowie deren Vor- und Nachteile im Hinblick auf Responsiveness, Usability und Erweiterungsfähigkeit.

Aufbauend auf dieser Analyse werden mehrere Verbesserungen vorgestellt, die in Rslidy v2 eingeführt wurden. Die Entwicklungsumgebung wurde durch einen überarbeiteten Build Prozess und eine verbesserte Abhängigkeitsverwaltung modernisiert. Ein neu gestaltetes Druckkonfigurationssystem ermöglicht es, Folienlayout, skalierung und Folienauswahl direkt innerhalb der Presentation über dynamisch generierte CSS Regeln zu steuern. Darüber hinaus verbessern Responsive Tabellen mit automatischen Datenbeschriftungen und interaktiver Sortierung die Darstellung strukturierter Daten auf unterschiedlichen Bildschirmgrößen.

Zusammen zeigen diese Beiträge, wie moderne Web Technologien eingesetzt werden können, um leichtgewichtige, aber leistungsfähige Foliensätze zu entwickeln. Das resultierende Framework verbessert Responsivität, Druckkonsistenz und Wartbarkeit, während der Browser-native Ansatz des ursprünglichen Rslidy Systems erhalten bleibt.

Contents

List of Figures

List of Tables

List of Listings

Acknowledgements

I would like to express my sincere gratitude to my family for their continuous support, encouragement, and understanding throughout my studies and during the completion of this thesis.

I am especially grateful to my advisor, Keith Andrews, for his guidance and support. His valuable feedback and the considerable time and effort he devoted to reviewing and proofreading this thesis were essential to its development and completion.

Fabian Platzer

Graz, Austria, 30 Jun 2026

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The thesis was written using the skeleton thesis provided by [Andrews 2021].
- Figures illustrating existing slide deck systems (e.g. Remark, Marp, Reveal.js, impress.js, Shower, Deck.js, and others) were created by the author using the respective slide deck tools [Bang 2023; Hattori 2026; El Hattab 2026; Szopka and Ingo 2025; Makeev 2026; Troughton 2016; Fu 2026a; Dalgleish 2020; Verou 2026].
- Screenshots of hosted slide deck platforms (e.g. Google Slides, Prezi, Mentimeter, Zoho Show, Visme, and Microsoft Sway) were taken from their publicly accessible web interfaces for demonstration and comparison purposes [Google 2026b; Halácsy et al. 2026; Warström et al. 2026; Zoho 2026; Taei 2026a; Microsoft 2026b].
- All figures and screenshots related to Rslidy v1 and Rslidy v2 were created by the author unless otherwise stated.

Chapter 1

Introduction

Over the past decade, the World Wide Web has evolved from a collection of static documents into a platform capable of running complex applications directly in the browser. Modern web technologies enable interactive user interfaces, responsive layouts, and data-driven visualisations without requiring native software installations. Consequently, the browser has increasingly become a universal runtime environment for many types of software, including presentation systems [Flanagan 2020, pages 2–3].

Traditional slide presentations are typically created using desktop applications such as Microsoft PowerPoint [Microsoft 2026a] or Apple Keynote [Apple 2026]. While these tools provide rich editing environments and visual features, they rely on proprietary file formats and are often tightly coupled to specific operating systems. Sharing and maintaining presentations across platforms can therefore introduce compatibility challenges and workflow limitations.

Web-based slide deck systems provide an alternative approach. Presentations are authored using open web technologies such as HTML, CSS, and JavaScript and rendered directly by the browser. This allows presentations to be shared easily, viewed on a wide range of devices, and extended with interactive behaviour or dynamic content. Over time, a diverse ecosystem of web-based slide frameworks has emerged, ranging from simple text-based tools to sophisticated hosted platforms with collaborative editing and analytics features.

Rslidy is a lightweight HTML-based slide framework whose first version (Rslidy v1) was developed by Patrick Hipp at Graz University of Technology as part of his Master's thesis [Hipp 2021b]. The framework demonstrates how presentations can be implemented entirely using open web technologies while remaining independent of external services or complex build environments.

Although Rslidy v1 provides a robust foundation for browser-based presentations, several limitations remain. In particular, aspects such as print configuration, responsive handling of complex slide elements, and maintainability of the development workflow reveal opportunities for further improvement. Addressing these challenges forms the motivation for the development of Rslidy v2, which is implemented and extended in this thesis.

1.1 Web-Based Slide Deck Systems

Web-based slide decks can be broadly divided into several categories. Some tools emphasise text-based authoring using formats such as Markdown, allowing slide content to be written quickly with minimal overhead. Other tools provide JavaScript libraries that extend standard HTML documents with slide navigation and presentation features. A third category consists of hosted presentation platforms, which combine visual editing tools with cloud-based collaboration and analytics.

Each of these approaches offers distinct advantages and limitations. Text-based systems are typically lightweight and easy to version control, but may provide fewer visual editing capabilities. Hosted

platforms enable collaborative workflows and integrated sharing, but rely on external infrastructure and may introduce performance or privacy issues. JavaScript-based frameworks often provide a balance between these two by combining flexibility with browser-native rendering. Rslidy is designed as a lightweight responsive HTML-based tool that allows presentations to be created and delivered entirely through the browser. This design emphasises simplicity, portability, and responsiveness, making it suitable for a wide range of presentation scenarios.

1.2 Motivation for Rslidy v2

While Rslidy v1 provides a stable foundation for HTML-based presentations, practical use reveals several limitations in both the development workflow and the runtime behaviour. Dependency management through `npm` [NPM 2026a] can result in redundant packages and non-deterministic builds, complicating reproducibility and long-term maintenance.

The printing process further depends on browser-specific rendering, which can lead to inconsistent margins, scaling issues, or truncated slides when exporting to PDF. Responsiveness is also constrained for complex structures such as wide tables or dense layouts, which often require manual adjustment to remain usable across different devices. In addition, lowlight mode requires manual activation and does not adapt to system-level or browser preferences.

These limitations motivated the development of Rslidy v2, which addresses these shortcomings through a revised and more consistent system design. The focus lies on improving maintainability, ensuring predictable print output, and strengthening responsive behaviour, while preserving the lightweight, browser-native approach of the original framework.

1.3 Scope and Structure

The objective of this Master's thesis is to analyse the current landscape of web-based slide deck tools and to extend the Rslidy framework with new features and architectural improvements. The work focuses on responsiveness, maintainability, and improved presentation of structured content within browser-based slide decks. The thesis is structured as follows:

- *Chapter 2: Modern Web Technologies*: Introduces the core technologies and concepts relevant to modern web development, including HTML and the Document Object Model (DOM), SVG, CSS, JavaScript and TypeScript, JavaScript Web APIs, build tools, and the principles of responsive web design.
- *Chapter 3: Web-Based Slide Decks*: Provides an overview of existing slide deck tools and compares their features, design approaches, and limitations.
- *Chapter 4: Rslidy v1*: Describes the architecture and capabilities of the original Rslidy v1 framework.
- *Chapter 5: Rslidy v2*: Presents the design and implementation of the improved framework and explains the major extensions introduced in this version.
- *Chapter 6: Selected Details of the Implementation*: Examines specific implementation aspects of Rslidy v2 in greater detail.
- *Chapter 7: Outlook and Future Work*: Outlines possible directions for further development, including deterministic export, speaker view support, slide virtualisation, remote control features, and automated testing.
- *Chapter 8: Concluding Remarks*: Summarises the main contributions of Rslidy v2 and reflects on the work carried out in this thesis.

Chapter 2

Modern Web Technologies

The core technologies and concepts relevant to modern web development encompass: HTML and the Document Object Model (DOM), SVG, CSS, JavaScript (JS) and TypeScript (TS), the JavaScript Web APIs, build tools, and the principles of responsive web design. To improve consistency and reliability across the web platform, Interop is a coordinated cross-browser effort [WPT 2026] to reduce behavioural differences and align implementations of existing specifications, with the goal that web technologies behave consistently across browsers. Furthermore, the Web Platform Baseline framework [web.dev 2026] provides a shared definition of feature availability across a core set of modern browsers: Chrome (desktop and Android), Firefox (desktop and Android), Safari (macOS and iOS), and Edge (desktop). With reference to a feature becoming Baseline, three stages of feature availability can be distinguished:

- *Limited Availability*: The feature does not yet work across the full set of core browsers.
- *Newly Available*: The feature works across the latest core browser versions, but may not be supported on older devices or browsers.
- *Widely Available*: The feature is well established, having worked across all core browsers for at least 30 months.

2.1 Modern HTML and the Document Object Model (DOM)

The HyperText Markup Language (HTML) is the standard markup language used to describe the structure and semantics of web content [WHATWG 2026]. Each HTML document is composed of elements organised hierarchically, defining headings, paragraphs, images, lists, and interactive components. HTML5 was standardised by the World Wide Web Consortium (W3C) in 2014 [W3C 2014], and the language was expanded with semantic elements such as `<header>`, `<article>`, and `<section>`, enabling developers to create more accessible and well-structured content. Nowadys, HTML has become a living standard, which is continually updated. It is maintained by the Web Hypertext Application Technology Working Group (WHATWG) [WHATWG 2026].

Beyond its structural role, modern HTML provides a range of built-in features that reduce the need for custom JavaScript and external libraries. Many of these features are now classified as Widely Available by the Web Platform Baseline initiative [web.dev 2026]. Native form controls include semantic input types such as `<input type="email">`, `<date>`, and `<range>`, which provide browser-level validation, accessibility support, and device-specific input optimisation without additional logic [MDN 2026g]. Structural disclosure elements such as `<details>` and `<summary>` enable built-in toggle behaviour with keyboard and screen-reader support, avoiding custom scripting for common UI patterns [MDN 2026a]. Media elements such as `<video>`, `<audio>`, and `<picture>` support native playback controls and responsive image selection, including resolution switching and art direction, and are now consistently interoperable across browsers [MDN 2026b].

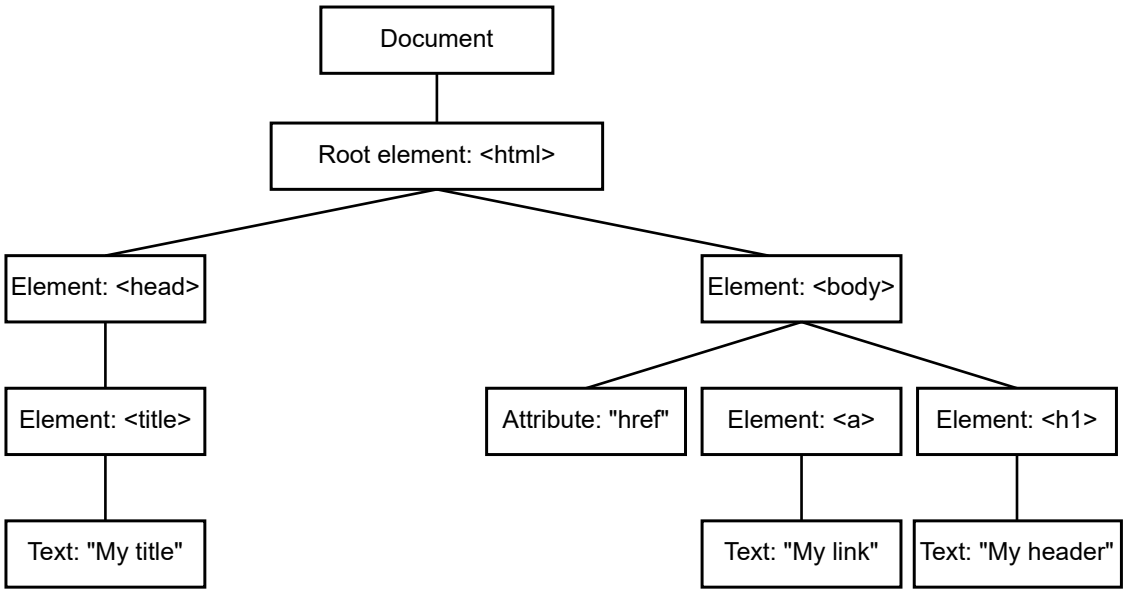


Figure 2.1: The Document Object Model (DOM) represents an HTML document as a hierarchical tree of objects, reflecting the structural relationships between elements.

The Document Object Model (DOM) provides a standardised way to access, manipulate, and update elements within a web document [W3C 2026]. Supported by all modern browsers, it represents the structure of a web page as a hierarchical tree, where each node corresponds to an element or text node, as shown in Figure 2.1. This tree and its parent and child elements can be accessed to locate nodes with specific attributes such as identifiers or class names.

There are many ways for developers to interact with the Document Object Model (DOM). Today, native web APIs such as the `getElementById()` and `addEventListener()` methods provide standardised and efficient means of accessing and manipulating the DOM without relying on external libraries. Modern JavaScript frameworks and libraries build upon these APIs to streamline event handling, state management, and rendering logic, while still adhering to the underlying DOM standards. An example demonstrating the use of the methods `getElementById()` and `addEventListener()` to interact with the Document Object Model (DOM) is shown in Listing 2.1.

2.2 Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) is an XML-based vector graphics format that is natively supported by modern browsers [MDN 2025e]. Unlike raster images, SVG graphics scale without loss of quality, which makes them suitable for responsive user interfaces, icons, diagrams, and annotations. SVG can be embedded in HTML as inline markup or referenced as an external resource. Inline SVG is part of the Document Object Model (DOM), allowing elements to be styled with CSS, addressed via selectors, and manipulated with JavaScript event listeners in the same manner as regular HTML content [MDN 2025d].

At time of writing, modern web browsers only fully support SVG 1.1 [W3C 2011]. There is only limited support for specific features of SVG 2, such as the styling of geometric properties via stylesheets, the `paint-order` property, and improved WAI-ARIA integration for accessibility. More specialised graphical additions from the SVG 2 specification, such as mesh gradients, hatch patterns, and the `z-index` property for SVG elements, remain largely unsupported across core browsers [W3C 2018].

From an implementation perspective, inline SVG is particularly useful when graphics must react to

```
// Access a DOM node by its identifier
const button = document.getElementById("submit-button");

// Register an event listener on the DOM node
if (button) {
  button.addEventListener("click", () => {
    console.log("Button clicked");
  });
}
```

Listing 2.1: Basic example demonstrating DOM access using `getElementById()`, and event handling with `addEventListener()`.

state changes, theming, or interaction. For example, CSS custom properties can be used to parameterise colours and stroke widths, while JavaScript can update attributes such as `viewBox` or path data to reflect application state. This integration supports interactive diagrams and UI controls without requiring canvas-based rendering or image regeneration.

SVG assets are commonly created using vector graphics editors, such as Inkscape [Inkscape 2026] or Adobe Illustrator [Adobe 2026], which allow diagrams and illustrations to be authored visually and exported as standards-compliant SVG files. These files can then be integrated directly into web documents or processed as part of a build workflow, preserving structural information such as paths, groups, and identifiers for further styling or scripting.

2.3 Modern CSS

Modern CSS extends responsive styling beyond viewport-based media queries by enabling component-scoped adaptation and more flexible conditional styling rules. A key addition are container queries, which allows descendant styles to depend on properties of a designated container rather than the global viewport [MDN 2026c]. Container queries are available in two variants [MDN 2026f]: size queries and style queries. Container size queries evaluate a container's dimensions, such as minimum or maximum inline or block size, aspect ratio, and orientation, allowing a component to adjust layout, typography, or spacing according to the space it occupies. Container style queries evaluate computed CSS values of the container, including custom properties, enabling conditional styling based on theme tokens or configuration variables without introducing additional classes or JavaScript.

Recent developments in Modern CSS reflect a broader shift towards moving layout logic, state handling, and interaction patterns from JavaScript into the browser's rendering engine. Features such as the relational selector `:has()`, native colour functions including `light-dark()` and `color-mix()`, and system-aware properties such as `accent-color` allow styles to react directly to document structure, user preferences, and rendering context. Scroll-linked animations based on `scroll-timeline` and `view-timeline` further enable visual state to be driven by scroll position without the need for scripting. This evolution is closely tied to the Interop initiative [WPT 2026], which has driven coordinated implementation of advanced CSS features across browser engines, while the Web Platform Baseline framework [web.dev 2026] provides a practical threshold for when these features can be used reliably in production. Together, they enable component-oriented and responsive styling based on native CSS capabilities, reducing runtime complexity and avoiding browser-specific workarounds.

2.4 JavaScript and TypeScript

JavaScript (JS) is a high-level programming language and core technology of the web, alongside HTML and CSS. It was originally developed by Brendan Eich in 1995 under the name LiveScript [Wirfs-Brock and Eich 2020, page 10]. Nowadays, JavaScript is standardised under the ECMAScript specification [ECMA 2026]. It defines the fundamental language constructs, object model, and execution semantics used in web applications [Flanagan 2020, page 2]. Over time, JavaScript has evolved from a simple scripting language into a versatile environment supporting both client-side and server-side development through platforms such as Node.js [Node 2026]. Since the release of ECMAScript 2015 (ES6), JavaScript has been significantly extended with modern language features, including block-scoped variables, arrow functions, native module support, and asynchronous programming using `async` and `await` [ECMA 2015]. These enhancements support the development of modular, maintainable, and scalable web applications.

TypeScript (TS), introduced by Microsoft in 2012, extends JavaScript with static typing, interfaces, and compile-time checks [Bierman et al. 2014]. TypeScript allows developers to describe data structures and function contracts explicitly, through selective type annotations and automatic type inference. TypeScript code transpiles to standard JavaScript, ensuring compatibility with all modern browsers and execution environments. By moving error checking from runtime to compile time, TypeScript improves code readability, supports refactoring in large codebases, and integrates closely with modern development tools such as static analysis and intelligent code completion [Baumgartner 2023]. It has therefore been widely adopted in frontend frameworks, such as Angular [Google 2026a], Vue.js [You 2026], and React [Meta 2026b].

2.5 JavaScript Web APIs

While the ECMAScript specification defines the JavaScript language itself, practical web applications rely on a wide range of runtime-provided interfaces known as JavaScript Web APIs. These APIs are implemented by browser engines and expose functionality for document interaction, user input, networking, asynchronous execution, storage, and access to selected device features [MDN 2025g]. Architecturally, Web APIs operate outside the JavaScript execution context. JavaScript code runs on a single main thread, while potentially blocking operations such as network requests, timers, or media processing are delegated to the browser. Results are delivered back to JavaScript via callbacks, promises, or event dispatching, allowing applications to remain responsive during asynchronous work.

Core Web APIs include the DOM and Events APIs for interacting with document structure and handling user input, forming the foundation of interactive user interfaces. Networking is primarily handled through the Fetch API, which provides a promise-based interface for HTTP requests and integrates naturally with modern asynchronous JavaScript patterns [MDN 2026d]. Timing-related APIs such as `setTimeout()` and `requestAnimationFrame()` support delayed execution and rendering-synchronised updates.

Client-side persistence is provided through storage APIs such as `localStorage`, `sessionStorage`, and IndexedDB, enabling applications to store state and cached data directly in the browser [MDN 2025b]. In addition, modern browsers expose APIs for media playback, device access, and fullscreen control, all governed by a strict, permission-based security model to protect user privacy and system resources.

2.6 JavaScript Module Systems: CJS, AMD, UMD, and ESM

In its early versions, JavaScript did not provide native language support for modular code organisation. As applications grew in size and complexity, this limitation made large-scale development challenging, since code reuse and encapsulation relied heavily on global variables or build-time concatenation of multiple source files. To overcome these constraints, four common modularisation patterns and module systems have emerged:

- *CommonJS (CJS)*: CJS is a synchronous, Node.js-oriented module system with runtime resolution and module caching [CommonJS 2026]. It is not natively supported in browsers and therefore requires bundling or transpilation for frontend use. CJS is still commonly found in existing backend codebases and development tooling.
- *Asynchronous Module Definition (AMD)*: AMD is an asynchronous module system designed primarily for browser environments [Garfield 2016]. It enables non-blocking module loading but relies on a verbose syntax and external loaders such as RequireJS. AMD has largely been superseded by ESM in modern frontend development.
- *Universal Module Definition (UMD)*: UMD is a hybrid wrapper that combines aspects of CJS and AMD to provide compatibility across both browser and server environments [UMD 2024]. UMD is mainly used when publishing libraries that must support a wide range of execution contexts.
- *ECMAScript Modules (ESM)*: ESM is the modern, standardised module system with native browser and Node.js support [ECMA 2026]. It enables static analysis, tree-shaking, and efficient bundling, and is suitable for both frontend and backend applications.

Each of these approaches defines its own syntax and loading semantics for importing and exporting functionality. An overview of the import and export syntax used by the different JavaScript module systems is shown in Listing 2.2.

ESM is now the standard module system in JavaScript [ECMA 2026]. It supports both synchronous and asynchronous loading and offers a clean, declarative syntax. ESM supports tree-shaking and static analysis, enabling more efficient bundling and optimisation. However, ESM code files must be served via an HTTP server rather than opened directly from the file system, since browsers block module imports in that context for security reasons [MDN 2026e]. A simple local web server can be created using Python and the following command:

```
python3 -m http.server 8000
```

2.7 Build Tools

As JavaScript applications grew in size and complexity, developers required tooling to automate repetitive tasks, manage dependencies, and prepare source code for deployment. Build tools address these needs by coordinating compilation, transformation, bundling, and optimisation steps within a reproducible workflow. In practice, two major categories can be distinguished: task runners and bundlers. While their responsibilities may overlap, they differ in focus and typical usage patterns.

2.7.1 Task Runners

Task runners provide a mechanism for defining and executing custom development tasks such as file transformation, asset copying, minification, and build orchestration. These tools focus on automating individual steps in a build workflow rather than analysing module dependencies or producing bundled output. Typical tasks handled by task runners include transpiling source code (for example from TypeScript to JavaScript), compiling stylesheets from preprocessors such as SCSS, optimising images, copying static assets to distribution directories, cleaning build artefacts, and triggering rebuilds in response to file system changes. Task runners are also commonly used to coordinate external tools, run tests, or execute deployment-related scripts as part of a unified workflow.

Gulp is a representative example of a task runner [Bublitz and Schoffstall 2026]. It uses a code-driven configuration model in which tasks are defined as JavaScript functions operating on file streams. This approach allows fine-grained control over build steps and enables developers to compose custom pipelines for processing JavaScript, CSS, images, or other assets. Task runners are particularly useful in projects

```
// -------------------------
// CommonJS (CJS)
// -------------------------
const util = require('./util.js');

module.exports = function process(data) {
  // perform operation
};

// -------------------------
// Asynchronous Module Definition (AMD)
// -------------------------
define(['./util'], function (util) {
  return function process(data) {
    // perform operation
  };
});

// -------------------------
// Universal Module Definition (UMD)
// -------------------------
(function (root, factory) {
  if (typeof define === 'function' && define.amd) {
    define(['./util'], factory);
  } else if (typeof module === 'object' && module.exports) {
    module.exports = factory(require('./util'));
  } else {
    root.myModule = factory(root.util);
  }
}(this, function (util) {
  return function process(data) {
    // perform operation
  };
}));

// -------------------------
// ECMAScript Modules (ESM)
// -------------------------
import { util } from './util.js';

export function process(data) {
  // perform operation
}

export default function main() {
  // entry point
}
```

Listing 2.2: The import and export syntax for CJS, AMD, UMD, and ESM module systems for JavaScript.

with specialised or non-standard build requirements, but they typically rely on additional tooling for dependency-aware bundling.

2.7.2 Bundlers

Bundlers focus on analysing module dependencies and combining multiple source files into optimised output bundles suitable for deployment. They reduce the number of required network requests, eliminate unused code, and apply transformations to ensure browser compatibility. Four of the most widely used bundlers are:

- *Webpack*: Webpack treats every file as a module and constructs a dependency graph starting from defined entry points [OJSF 2026]. Transformation steps are handled through loaders, while plugins extend the build process with tasks such as code optimisation, asset generation, and environment-specific configuration. A simple Webpack configuration is shown in Listing 2.3.
- *Rollup*: Rollup is primarily designed for bundling JavaScript libraries and leverages the static structure of ESM modules to enable effective tree-shaking [Rollup 2026]. By analysing imports and exports at build time, Rollup can eliminate unused code and generate compact output bundles in multiple formats.
- *Parcel*: Parcel emphasises ease of use through a zero-configuration approach [Govett 2026]. It automatically detects dependencies and applies appropriate transformations without requiring explicit configuration. This makes it suitable for rapid development and prototyping.
- *Vite*: Vite is a modern bundler built around native ESM modules [Vite 2026]. During development, it serves source files directly to the browser using ESM, resulting in fast startup times and efficient hot-module replacement. For production builds, Vite relies on Rollup to produce optimised bundles, combining fast development workflows with efficient deployment output.

All of the above bundlers support tree-shaking for ESM modules, eliminating unused code in production builds. While Rollup treats it as a core design principle, Webpack, Parcel, and Vite achieve the same result through static ESM analysis, differing mainly in their configuration approaches and default optimisation behaviours.

2.8 Responsive Web Design (RWD)

Responsive Web Design (RWD) represents a fundamental design approach to modern web development, enabling web content to adapt seamlessly to a wide range of devices, screen sizes, and interaction contexts. Instead of maintaining separate layouts for desktops, laptops, tablets, and smartphones, responsive design allows a single application to dynamically adjust its structure and presentation based on the available viewport and device characteristics [MDN 2025c].

A central technique in responsive web design is the use of flexible layouts, which allow layout elements to scale proportionally, by using relative units such as percentages or viewport-based measurements instead of fixed pixel values. Fluid images ensure that visual assets resize smoothly within their containers to avoid overflow or distortion. Media queries provide conditional styling rules that enable layouts to respond to screen size, orientation, output medium (such as print), and other device properties [Marcotte 2010; Marcotte 2015].

Modern CSS has significantly extended these foundational principles through advanced layout and styling mechanisms. Layout systems such as Flexbox and CSS Grid provide declarative, one- and two-dimensional control over component alignment and distribution, reducing the need for rigid layout assumptions [Meyer and Weyl 2023]. Container size queries further refine responsive design by allowing components to adapt based on the size of their parent container rather than the global viewport. In

```
// webpack.config.js
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
        },
      },
    ],
  },
};
```

Listing 2.3: Example of a basic Webpack configuration file defining entry points, output paths, and loaders, adapted from the official Webpack documentation [OJSF 2026].

addition, relative CSS length units such as `em`, `rem`, `vw`, and `vh` enable layouts and typography to scale in relation to font size or viewport dimensions, supporting responsive designs that adapt naturally across different devices and display contexts [MDN 2025a].

By combining these techniques, web applications can react to changes in the browser environment and adjust both layout and presentation in real time. This adaptability improves usability, accessibility, and visual consistency across devices.

Chapter 3

Web-Based Slide Decks

The evolution of web-based slide decks has significantly transformed the way presentations are created and shared. The pioneering tools in this space were Slidy [Raggett 2005] and Slidy2 [Raggett 2006], which introduced the idea of HTML-based slide decks as an alternative to traditional tools such as Microsoft PowerPoint [Microsoft 2026a] and Apple Keynote [Apple 2026].

This chapter is based on a previous survey by the auther of this thesis [Platzer 2024], which itself builds upon Patrick Hipp's comprehensive 2019 survey [Hipp 2019]. The web-based slide decks in this survey are grouped into four categories: text-based, JavaScript-based, hosted, and responsive. Each category is described in a dedicated section.

To better compare the tools, a listing and a figure of the same example slide were created with every tool at the same window size (1600×900). The example slide has three bullet points on the left and a vector graphic (in SVG format) on the right. If an SVG graphic cannot be displayed by a particular tool, a PNG image is used instead. In most cases, custom CSS styling is used to achieve a two-column slide layout. Where the CSS styling is too extensive, it has been omitted from the example listings for brevity. All the slide deck tools are compared across the 17 criteria shown in Table 3.1.

3.1 Early Systems

The earliest innovations in the field of web-based slide decks were Slidy and its successor Slidy2. These tools marked the beginning of a transition away from traditional desktop presentation software towards more flexible, web-based solutions. This section introduces both systems and their respective contributions to online slide presentation frameworks.

3.1.1 Slidy [2005-2006]

Slidy is an HTML-based slide presentation tool developed by Dave Raggett [Raggett 2005], a member of the W3C team and a pioneer in web standards. Slidy was designed to create slide shows that could be viewed directly in web browsers. It generates a slide-show from a single XHTML file, which includes a CSS file for styling and a JavaScript file for the functionality. Each slide is enclosed in a `<div class ="slide">` element. An example can be seen in Figure 3.1; the corresponding source code is shown in Listing 3.1.

The key features of Slidy include:

+ *SVG Inclusion*: Include vector graphics as SVG.

+ *Incremental Display*: Bullet items revealed incrementally.

Feature	Description
Collaboration	Whether the tool allows multiple users to work on a presentation simultaneously.
SVG Inclusion	Whether the slide deck tool supports inclusion of SVG graphics.
Globally Change Bullet Spacing	Ability to modify the vertical spacing between bullet points throughout the whole presentation.
Globally Change Bullet Indentation	Ability to adjust bullet point indentation across all slides.
Export as PDF	Whether the tool supports exporting the presentation as a PDF file.
Export as HTML	Whether the tool supports exporting the presentation in an HTML format.
Export as PowerPoint	Ability to export the presentation in Microsoft PowerPoint format.
Slide Numbering	Support for automated slide numbering in the presentation.
Live Code Integration	Support for embedding of live code within the presentation.
Open Source	Whether the tool is open-source, allowing for customisation and modification.
Licence	The licensing model, indicating whether the tool is proprietary or open for public use.
First Release	The first known release of a tool.
Last Update	Most recent update of the tool, reflecting its ongoing development.
Popularity (Usage)	The level of popularity and usage of the tool. Measured by GitHub stars where appropriate, otherwise estimated.
Swipe Navigation	Whether the tool supports swiping gestures to navigate forwards and backwards between slides.
Margin Navigation	Whether the tool supports tapping or clicking the slide margins to navigate forwards and backwards between slides.
Figure Zooming and Panning	The ability to zoom in and out and pan around in embedded figures (images and graphics).
Presenter Mode	Whether the slide deck tool offers a presenter mode with a second window for speaker notes.

Table 3.1: Comparison criteria for the slide deck tools.

+ *Keyboard Shortcuts*: Easy slide navigation during presentations.
+ *Based on HTML and CSS*: Easy customisation and styling using standard web technologies.
+ *JavaScript-Enhanced*: Interactive features like navigation controls, slide transitions, and multimedia embedding.

Slidy's use of HTML and CSS for structure and styling made it highly appealing to web developers.

3.1.2 Slidy2 [2006-2013]

Building on the foundation laid by Slidy, Slidy2 introduced several enhancements to improve usability and functionality [Raggett 2006]. The key enhancements include:

+ *Improved User Interface*: Improved navigation controls and user-friendly interface elements.
+ *Advanced Features*: Support for more complex animations, better multimedia integration, and responsive design for desktop screen sizes.

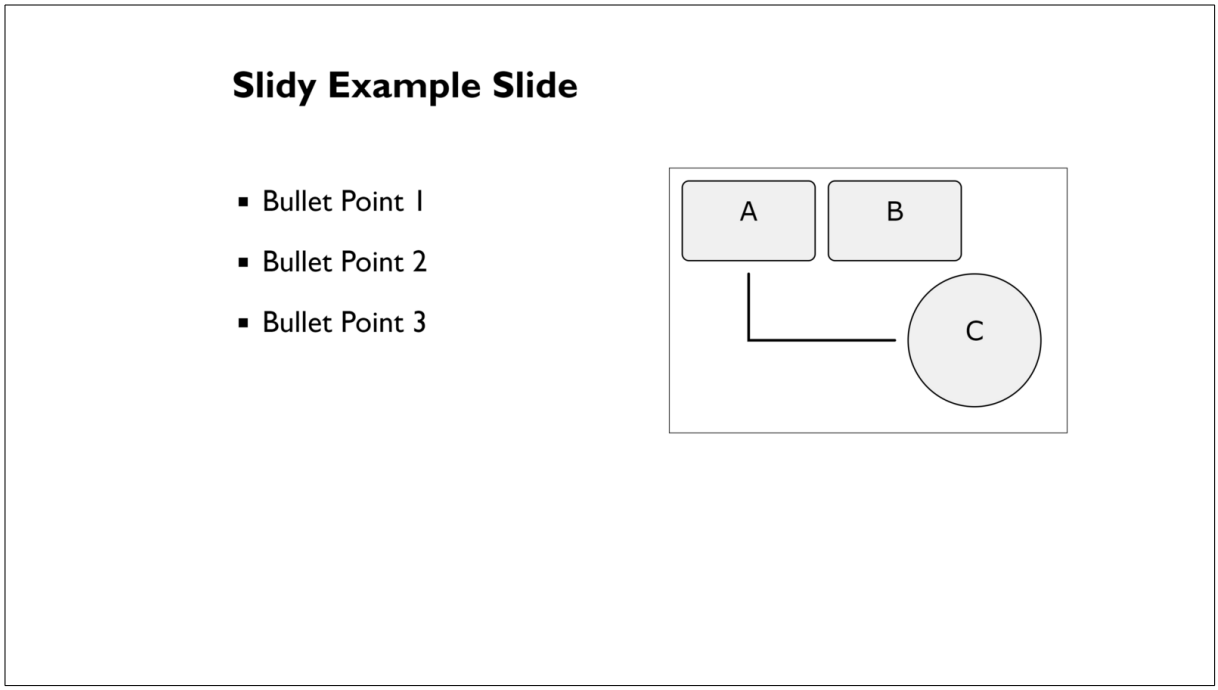


Figure 3.1: Slidy: Example slide. The corresponding source code is shown in Listing 3.1.

+ *Customisation Options*: Greater flexibility in customising the appearance and behaviour of slide decks through extended CSS and JavaScript support.

Although Slidy2's enhancements addressed some of the limitations of the original Slidy, the design was still optimised for desktop usage, which has its own problems and reduced functionality when used on mobile devices with smaller screens.

3.2 Text-Based Slide Decks

Text-based slide decks provide an efficient and user-friendly alternative to traditional presentation tools. At the core of many text-based slide deck tools is Markdown [Cone 2026], a lightweight markup language first proposed by Gruber [2004]. Markdown allows users to write content in plain text, using an easy-to-read, easy-to-write format, which is then converted to structurally valid XHTML (or HTML). Using Markdown requires little technical expertise. Users can focus on content creation, without needing deep knowledge of HTML, CSS, or JavaScript.

This section describes four popular tools, created in a Node-based environment [Node 2026], for building text-based slide decks using Markdown: Remark, Marp, Fusuma, and Slidev. Other systems, like Slide Show (S9) [Bauer 2017], Slidedown [Nakajima and Croak 2012], and Slidifier [Ludvigsen and Halvorsen 2016], were once widely used, but are now considered outdated. This survey does not include these systems, but refers the reader back to the survey by Hipp [2019].

3.2.1 Remark [2011-2023]

Remark is a simple, browser-based slide deck tool designed for users familiar with HTML and CSS, integrating both HTML and Markdown [Bang 2023]. To get started with Remark, the user needs to create an HTML file for the slide deck, or use the provided boilerplate template, open it in a browser, and edit the Markdown or CSS as needed. To enhance the experience, keyboard shortcuts, such as "C" to clone a display, "P" to switch to presenter mode, and "H" to access the help menu, are available. In a Remark HTML

```
<div class="slide">
  <h1>Slidy Example Slide</h1>
  <div class="container">
    <div class="left-column">
      <ul>
        <li>Bullet Point 1</li>
        <li>Bullet Point 2</li>
        <li>Bullet Point 3</li>
      </ul>
    </div>
    <div class="right-column">
      <img src="graphics/diagram.svg" alt="Example Image">
    </div>
  </div>
</div>

<style>
  .container {
    display: flex;
    justify-content: space-between;
    align-items: flex-start;
  }
  .left-column {
    width: 45%;
  }
  .right-column {
    width: 45%;
  }
  img {
    max-width: 75%;
    height: auto;
    margin-top: 3rem;
  }
</style>
```

Listing 3.1: Slidy: Example slide source code. The resulting slide is shown in Figure 3.1.

file, style definitions are placed in the HTML header. The body includes a `<textarea id="source">` element, where the Markdown text or HTML for the slides is placed. Calling the `create()` function triggers the creation of a new slide deck, as can be seen in Figure 3.2, the corresponding source code is shown in Listing 3.2.

The key features of Remark include:

+ *SVG Inclusion*: Supports the import of SVG vector graphics.
+ *Export as PDF*: Supports exporting the presentation as PDF.
+ *Export as HTML*: Supports exporting as HTML.
+ *Slide Numbering*: Supports slide numbering.
+ *Swipe Navigation*: Supports touch gestures on smartphones and tablets.
+ *Margin Navigation*: Supports margin tapping or clicking for navigation.

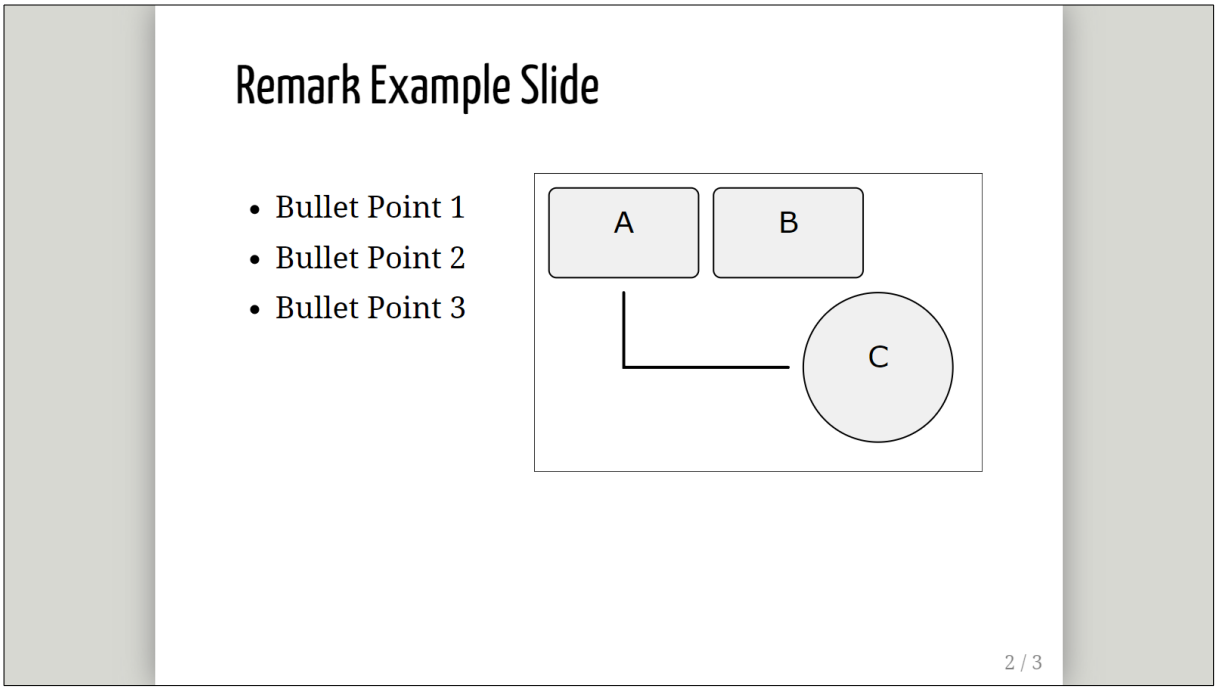


Figure 3.2: Remark: Example slide. The corresponding source code is shown in Listing 3.2.

\+ *Presenter Mode*: Features speaker notes and a cloned slideshow view for presentations.

Limitations of Remark include:

- *Collaboration*: No built-in support for collaboration.
- *Globally Change Bullet Spacing*: No built-in mechanism for global bullet spacing changes.
- *Globally Change Bullet Indentation*: No built-in mechanism for global bullet indentation changes.
- *Export as PowerPoint*: Not possible to export as PowerPoint.
- *Live Code Integration*: Does not offer live code integration.
- *Figure Zooming and Panning*: Lacks built-in functionality for figure zooming and panning.

3.2.2 Marp [2018-]

Marp combines the simplicity of Markdown with the flexibility of HTML, offering features such as slide-specific styles, presenter notes, and a presenter view [Hattori 2026]. Slides are separated using three dashes "`---`", and presenter notes are added with XML comments enclosed in "`<!--`" and "`-->`".

Marp's input format is an extension of a standard Markdown variant called CommonMark [MacFarlane et al. 2026]. A Marp input file starts with YAML front matter enclosed by three dashes "`---`", which allows for global slide settings such as themes and transitions. The body of the document contains the Markdown text for the slides. HTML elements can also be mixed in. A simple example can be seen in Figure 3.3; the corresponding source code is shown in Listing 3.3. Marp can then be run on the command line with the command:

```
marp deck.md
```

or as a plugin for Visual Studio Code.

The key features of Marp include:

```
<body>
<textarea id="source">
---
# Remark Example Slide

.left-column[
* Bullet Point 1
* Bullet Point 2
* Bullet Point 3
]

.right-column[
  ![Image Description](./diagram.svg)
]

---
</textarea>
<script src="https://remarkjs.com/downloads/remark-latest.min.js">
</script>
<script>
    var slideshow = remark.create();
</script>
</body>

<style>
.left-column{
  width: 40%;
  float: left;
  line-height: 3rem;
  font-size: 2rem;
}

.right-column{
  width: 60%;
  float: right;

}
#img {
  width: 12.5rem;
  height: 6rem;
}
</style>
```

Listing 3.2: Remark: Example slide source code, created with the `boilerplate.html` template from GitHub [Bang 2023]. The Markdown to create the slide deck must be inside a `<textarea>` element. Calling the `create()` function creates a new slideshow. The resulting slide is shown in Figure 3.2.

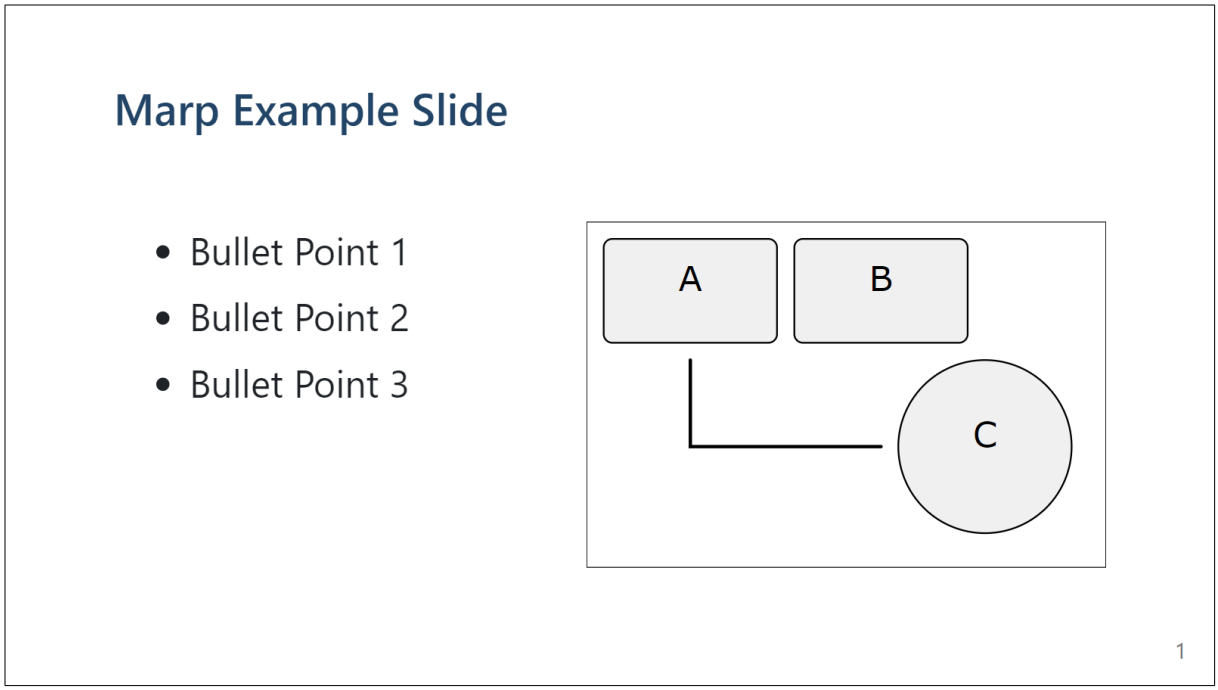


Figure 3.3: Marp: Example slide. The corresponding source code is shown in Listing 3.3.

+ *SVG Inclusion*: Supports the import of SVG vector graphics.
+ *Globally Change Bullet Spacing*: Bullet spacing can be adjusted globally through themes and global styles.
+ *Globally Change Bullet Indentation*: Bullet indentation can be adjusted globally through themes and global styles.
+ *Export as PDF*: Supports exporting as PDF.
+ *Export as HTML*: Supports exporting as HTML.
+ *Export as PowerPoint*: Supports exporting as PowerPoint.
+ *Slide Numbering*: Supports slide numbering.
+ *Presenter Mode*: Presenter view with speaker notes.

Limitations of Marp include:

- *Collaboration*: No built-in support for collaboration.
- *Live Code Integration*: Does not offer live code integration.
- *Swipe Navigation*: No swipe gestures for slide navigation.
- *Margin Navigation*: No margin tapping or clicking for slide navigation.
- *Figure Zooming and Panning*: Lacks built-in functionality for figure zooming and panning.

```
---
marp: true
theme: default
paginate: true
backgroundColor: #fff
---

# Marp Example Slide
<!-- _class: split -->

<div class=ldiv>
- Bullet Point 1
- Bullet Point 2
- Bullet Point 3
</div>

<div class=rdiv>
![Example Image](./diagrams/diagram.svg)
</div>

<style>
  section.split {
  overflow: visible;
  display: grid;
  grid-template-columns: 31rem 34rem;
  grid-template-rows: 9rem auto;
  grid-template-areas:
      "slideheading slideheading"
      "leftpanel rightpanel";
  }

  section.split h3,
  section.split .ldiv,
  section.split .rdiv { border: 0px}
  section.split h3 {
      grid-area: slideheading;
      font-size: 3rem;
  }
  section.split .ldiv { grid-area: leftpanel; font-size: 2.5rem;}
  section.split .rdiv { grid-area: rightpanel; }
</style>
```

Listing 3.3: Marp: Example slide source code. Custom CSS styling is used to achieve the two-column layout for the example slide. The resulting slide is shown in Figure 3.3.

3.2.3 Fusuma [2018-2021]

Fusuma is a text-based online slide deck tool for Node [Node 2026], which allows users to create web-based presentations by editing plain text or Markdown files [Hiroto 2021]. Fusuma supports MDX [MDX 2026], which is a format that lets the user use JSX [Meta 2026a] in Markdown documents. Fusuma can be installed over the command line with the command:

```
npm i fusuma -D
```

The command:

```
npx fusuma init
```

creates the project structure and the server can then be run with the command:

```
npx fusuma start
```

As in many other Markdown slide deck tools, slides in Fusuma are divided by three "`---`". Speaker notes can be added with "`<!-- note`", these can then be seen in the presenter mode. An example slide can be seen in Figure 3.4, with the corresponding source code in Listing 3.4.

The key features of Fusuma include:

+ *SVG Inclusion*: Supports the import of SVG vector graphics.
+ *Export as PDF*: Supports exporting as PDF.
+ *Export as PowerPoint*: Supports exporting as PowerPoint.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Margin Navigation*: Supports margin tapping for navigation.
+ *Presenter Mode*: Presenter mode with speaker notes.

Limitations of Fusuma include:

- *Collaboration*: No built-in support for collaboration.
- *Globally Change Bullet Spacing*: No built-in mechanism for global bullet spacing changes.
- *Globally Change Bullet Indentation*: No built-in mechanism for global bullet indentation changes.
- *Export as HTML*: Does not support HTML export.
- *Slide Numbering*: No automated slide numbering.
- *Live Code Integration*: Does not offer live code integration.
- *Figure Zooming and Panning*: Lacks built-in functionality for figure zooming and panning.

3.2.4 Slidev [2021-]

Slidev is a modern, web-based presentation tool using Markdown [Fu 2026a]. It stands out for its extensive feature set, flexibility, and interactivity through native support for Vue.js [You 2026] components, which can be directly embedded into slides. As an open-source tool with an active community, it benefits from continuous maintenance and feature development. Slide content is written in Markdown and organised into individual slides separated by three dashes "`---`" placed on a separate line. Each slide is internally rendered as a Vue component, which allows the integration of reactive elements, custom layouts, and client-side logic. In addition to standard Markdown syntax, Slidev supports front matter configuration at

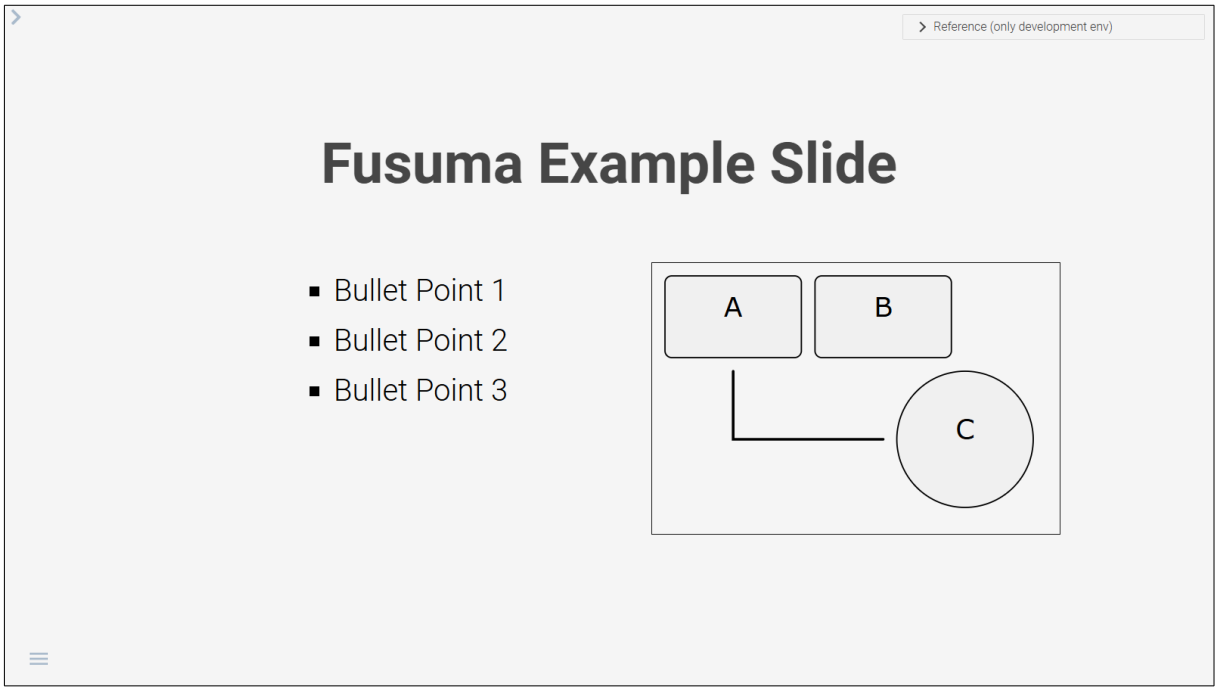


Figure 3.4: Fusuma: Example slide. The corresponding source code is shown in Listing 3.4.

```
---
Fusuma Example Slide
---
<div style={{ display: "flex", justifyContent: "space-between"}}>

<div style={{ width: "50%" }}>
  - Bullet Point 1
  - Bullet Point 2
  - Bullet Point 3
</div>

<div style={{ width: "40%" }}>
  <img src="./diagram.svg" alt="Example Diagram"/>
</div>

</div>
```

Listing 3.4: Fusuma: Example slide source code. Written in Markdown with CSS styling to achieve two columns for the example slide. The resulting slide is shown in Figure 3.4.

both global and per-slide level, enabling the definition of layouts, transitions, themes, and meta settings. Layouts such as `two-cols` provide predefined structural patterns, while the special `::right::` syntax enables structured content placement within multi-column layouts. An example Slidev slide can be seen in Figure 3.5, with the corresponding source code in Listing 3.5.

A Slidev slide deck can be built locally with npm, pnpm, or yarn:

```
npm init slidev@latest
npm run dev
npm run build
```

Slidev can also be started directly in the browser using StackBlitz [Fu 2026b; Fu 2026c].

A slide deck can be exported to PDF with the command:

```
npm run export
```

and to PowerPoint and PNG with the command:

```
npm run export --format [format]
```

The key features of Slidev include:

+ *SVG Inclusion*: Supports the import of SVG vector graphics.
+ *Globally Change Bullet Spacing*: Bullet spacing can be adjusted globally through themes and global styles.
+ *Globally Change Bullet Indentation*: Bullet indentation can be adjusted globally through themes and global styles.
+ *Export as PDF*: Supports exporting as PDF.
+ *Export as PowerPoint*: Supports exporting as PowerPoint.
+ *Slide Numbering*: Supports slide numbering.
+ *Live Code Integration*: Supports embedding of live code blocks and interactive code demos.
+ *Open Source*: Open-source project with an active community.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Margin Navigation*: Supports margin tapping or clicking for navigation.
+ *Presenter Mode*: Presenter view with speaker notes.

Limitations of Slidev include:

- *Collaboration*: No built-in support for collaboration.
- *Export as HTML*: Not possible to export presentations as standalone HTML files.
- *Figure Zooming and Panning*: Lacks built-in functionality for figure zooming and panning.

3.2.5 Comparison of Text-Based Slide Decks

A comparison of text-based slide deck tools is shown in Table 3.2. Marp, Fusuma, Remark, and Slidev, all support the inclusion of SVG graphics and exporting the slide deck to PDF. While only Marp and Remark offer HTML export, Marp, Fusuma, and Slidev offer the export of the slide deck as a PowerPoint

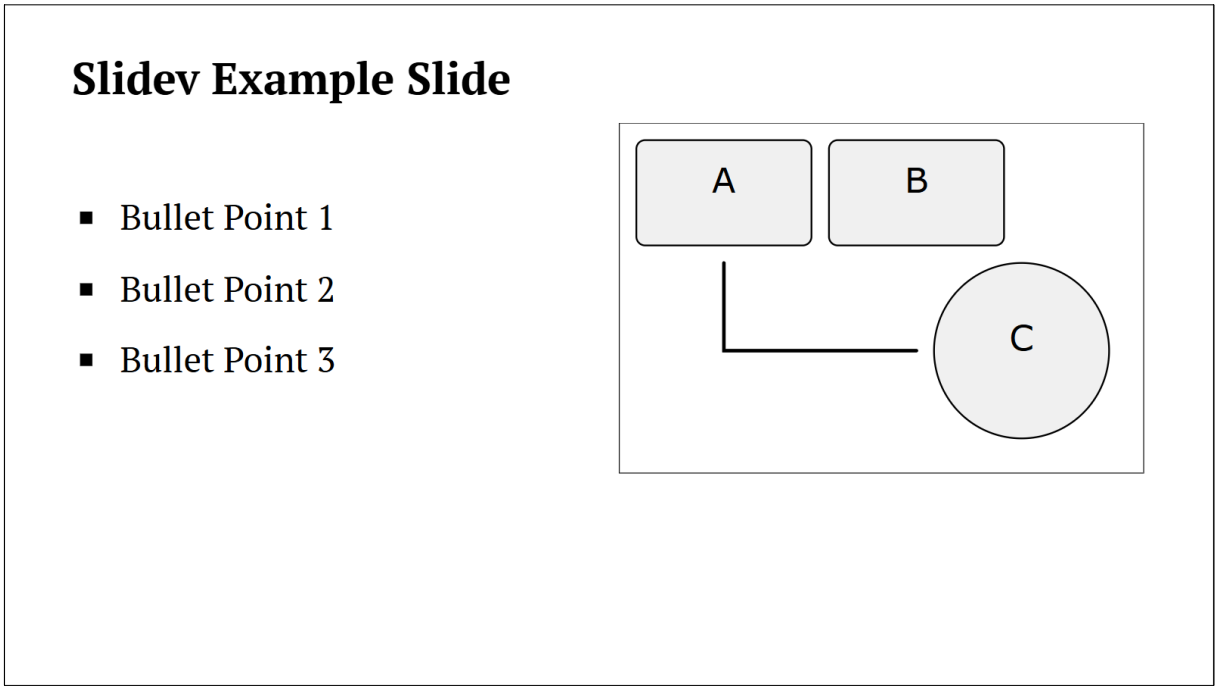


Figure 3.5: Slidev: Example slide. The corresponding source code is shown in Listing 3.5.

```
---
layout: two-cols
---
# Slidev Example Slide
- Bullet Point 1
- Bullet Point 2
- Bullet Point 3

::right::
#
<img src="diagrams/diagram.svg" alt="Example SVG">
```

Listing 3.5: Slidev: Example slide source code written in Markdown. The layout `two-cols` separates the page content into two columns. The resulting slide is shown in Figure 3.5.

Feature	Remark	Marp	Fusuma	Slidev
Collaboration:	✗	✗	✗	✗
SVG Inclusion:	✓	✓	✓	✓
Globally Change Bullet Spacing:	✗	✓	✗	✓
Globally Change Bullet Indentation:	✗	✓	✗	✓
Export as PDF:	✓	✓	✓	✓
Export as HTML:	✓	✓	✗	✗
Export as PowerPoint:	✗	✓	✓	✓
Slide Numbering:	✓	✓	✗	✓
Live Code Integration:	✗	✗	✗	✓
Open Source:	✓	✓	✓	✓
Licence:	MIT	MIT	MIT	MIT
First Release:	2011-10-15	2018-03-25	2018-04-27	2021-04-12
Last Update:	2023-06-27	2025-08-06	2021-12-01	2026-01-07
Popularity (GitHub stars):	13k	10.2k	5.4k	43.5k
Swipe Navigation:	✓	✗	✓	✓
Margin Navigation:	✓	✗	✓	✓
Figure Zooming and Panning:	✗	✗	✗	✗
Presenter Mode:	✓	✓	✓	✓

Table 3.2: Comparison of text-based slide deck tools.

presentation. However, these tools lack stand-alone collaborative capabilities and only offer limited customisation options. Global changes to bullet indentation and spacing can only be done with Marp and Slidev. This can be achieved with styling in a CSS file or the file itself. Only Slidev provides live coding integration, making it particularly useful for dynamic code presentations.

All tools are open-source with MIT licenses and have varying levels of popularity, with Slidev being the most popular. Only Marp fails to support swipe and margin tap navigation. All of the tools provide a presenter mode with a second window for speaker notes, enhancing the convenience for presenters. Compared to JavaScript-based slide decks, these tools are easy to set up with Markdown syntax and provide many key features for presentations, like easy export to PDF and some user-friendly but limited customisation of slides.

3.3 JavaScript-Based Slide Decks

JavaScript-based slide deck tools have transformed the way presentations are created by using the power of HTML, CSS, and JavaScript to create interactive, and customisable slides with animations, and interactions that are not typically available in static or traditional slide tools. These presentations are rendered in the browser, making them accessible from various devices with internet access. Most JavaScript-based slide deck tools are available as packages on NPM, a package manager which allows developers to manage and install JavaScript packages for Node projects [NPM 2026a].

This section covers six popular tools: Shower, Reveal.js, Deck.js, impress.js, Bespoke.js, and Inspire.js. All of these tools are Node-based, except for Deck.js. The tools are compared based on specific criteria, showing insights into their features, functionality, and overall effectiveness for creating JavaScript-based presentations.

3.3.1 Shower [2010-2024]

Shower is a minimalistic HTML presentation tool that focuses on providing essential features without overwhelming users with complex options and features [Makeev 2026]. It imitates the appearance of traditional slide decks like PowerPoint. Slides are created with an `<section class="slide">` element, as shown in Figure 3.6 and the corresponding Listing 3.6. Presentations can be hosted on Netlify, a web hosting platform that simplifies building, deploying, and managing static websites web applications directly from a Git repository [Netlify 2026], as well as self-hosted with NPM. A gulp configuration file has predefined tasks such as `prepare`, which builds the slide deck, and `publish`, which uploads it to Netlify.

They key features of Shower include:

+ *SVG Inclusion*: Supports SVG graphics through native HTML embedding.
+ *Globally Change Bullet Spacing*: Global bullet spacing can be adjusted in shared CSS styles.
+ *Globally Change Bullet Indentation*: Global bullet indentation can be adjusted in shared CSS styles.
+ *Export as PDF*: Presentations can be exported to PDF.
+ *Export as HTML*: Presentations are deployed as HTML.
+ *Slide Numbering*: Supports slide numbering.
+ *Open Source*: Open-source project.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Margin Navigation*: Supports margin tapping or clicking for navigation.
+ *Presenter Mode*: Presenter mode with speaker notes.

Limitations of Shower include:

- *Collaboration*: No built-in support for collaboration.
- *Export as PowerPoint*: Not possible to export as PowerPoint.
- *Live Code Integration*: Does not support live code integration.
- *Figure Zooming and Panning*: Does not provide built-in zoom or pan features.

3.3.2 Deck.js [2011-2016]

Deck.js is a flexible and extensible JavaScript library designed for creating HTML-based presentations, based on jQuery and Modernizr [Troughton 2016]. It provides templates and themes to build a simple standard slide deck. The framework is not Node-based and includes a file named `boilerplate.html` with all the necessary extensions included. This file serves as a starting point and users can immediately edit slides in this file and view them on a web browser. The basic functionality is provided by the `deck.core` module, which provides the functionality for creating and moving through a slide deck and defines various states for the slide deck that can be customised using CSS. Additional features are available through extensions and plugins. A simple example can be seen in Figure 3.7. The corresponding source code is shown in Listing 3.7.

The key features of Deck.js include:

+ *SVG Inclusion*: Supports SVG graphics through native HTML embedding.
+ *Globally Change Bullet Spacing*: Global bullet spacing can be adjusted in shared CSS styles.

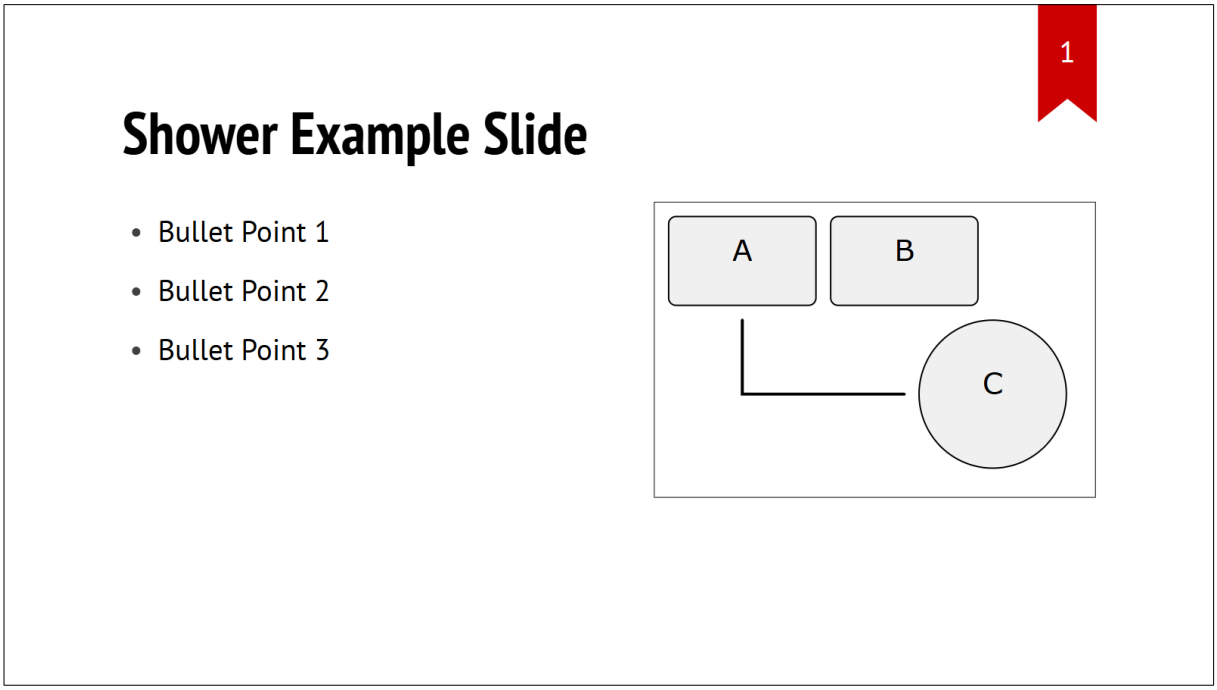


Figure 3.6: Shower: Example slide. The corresponding source code is shown in Listing 3.6.

```
<section class="slide">
  <h2>Shower Example Slide</h2>
  <div class="columns two">
    <ul>
        <li>Bullet Point 1</li>
        <li>Bullet Point 2</li>
        <li>Bullet Point 3</li>
    </ul>
    <div class="image">
        <img src="diagrams/diagram.svg" alt="Example Image">
    </div>
  </div>
</section>
```

Listing 3.6: Shower: Example slide source code. The resulting slide is shown in Figure 3.6.

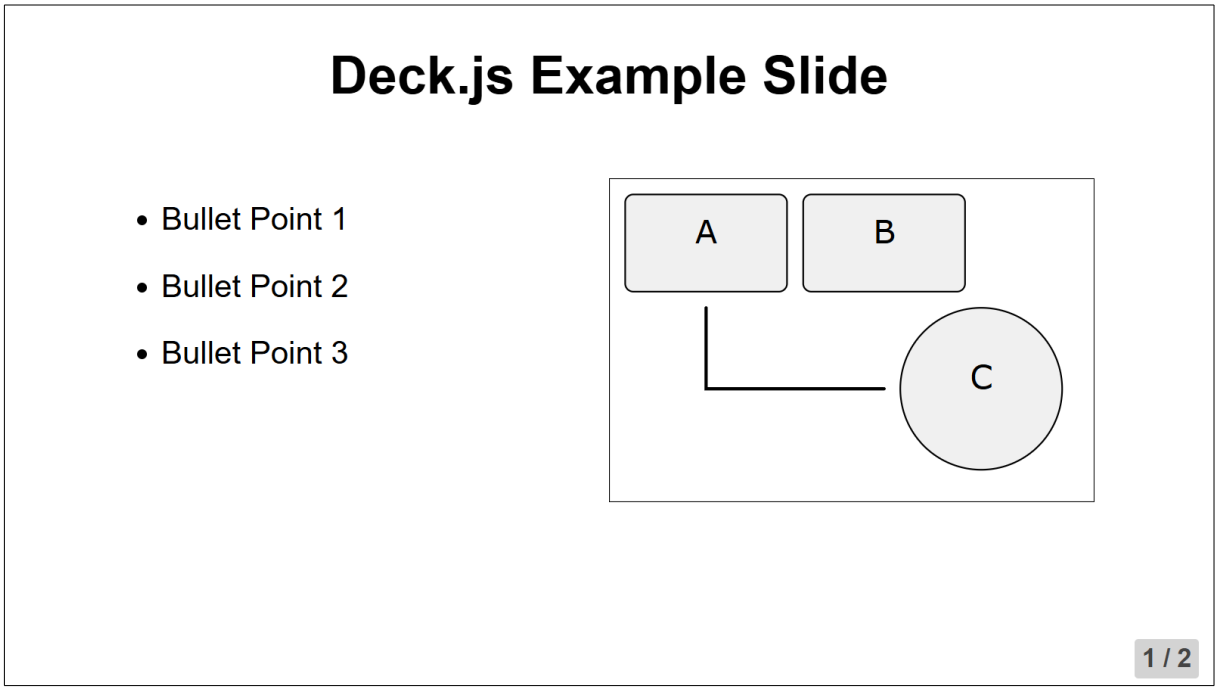


Figure 3.7: Deck.js: Example slide. The corresponding source code is shown in Listing 3.7.

+ *Globally Change Bullet Indentation*: Global bullet indentation can be adjusted in shared CSS styles.
+ *Export as HTML*: Presentations are deployed as HTML.
+ *Slide Numbering*: Supports slide numbering through extensions.
+ *Open Source*: Open-source project.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Margin Navigation*: Supports margin navigation.
+ *Presenter Mode*: Presenter mode through extensions.

Limitations of Deck.js include:

- *Collaboration*: No built-in support for collaboration.
- *Export as PDF*: Does not support exporting to PDF.
- *Export as PowerPoint*: Not possible to export as PowerPoint.
- *Live Code Integration*: Does not support live code integration.
- *Figure Zooming and Panning*: Does not provide built-in zoom or pan features.
- *Limited Updates*: Last update was in 2016, which may indicate limited support or feature advancements.

3.3.3 Reveal.js [2011-]

Reveal.js is an open-source JavaScript library for creating modern, web-based presentations [El Hattab 2026]. Like most other JavaScript-based slide deck tools, it allows users to create interactive slide decks

```
<div class="slide">
  <h3 class="title">Deck.js Example SLide</h3>
  <div class="slide-content">
    <div class="bullets">
      <ul>
        <li>Bullet Point 1</li>
        <li>Bullet Point 2</li>
        <li>Bullet Point 3</li>
      </ul>
    </div>
    <div class="image">
      <img src="diagrams/diagram.svg" alt="Example Image">
    </div>
  </div>
</div>

<!-- Required JS files. -->
<script src="jquery.min.js"></script>
<script src="core/deck.core.js"></script>

<!-- Some Extensions for Deck.js -->
<script src="extensions/status/deck.status.js"></script>


<!-- Initialise the deck. -->
<script>
  $(function() { $.deck('.slide');});
</script>
```

Listing 3.7: Deck.js: Example slide source code. Slides are created with the class `slide`, usually either `<section class="slide">` or `<div class="slide">`. Deck.js offers various extensions, like the status extension, which displays the slide numbers on the example slide. The slide deck is initialised with the `deck('.slide')` function. The resulting slide is shown in Figure 3.7.

using HTML, while enhancing them with CSS and JavaScript for smooth transitions and effects. Its basic features can be seen in Figure 3.8 and Listing 3.8. Advanced usage with different plugins can be seen in the demo version on GitHub [El Hattab 2026]. Additionally, slide decks using Reveal.js can be created and published via an online platform called Slides [El Hattab and Bossola 2026], which offers a fully-featured visual editor.

They key features of Reveal.js include:

+ *SVG Inclusion*: Supports SVG graphics through native HTML embedding.
+ *Globally Change Bullet Spacing*: Global bullet spacing can be adjusted in shared theme styles.
+ *Globally Change Bullet Indentation*: Global bullet indentation can be adjusted in shared theme styles.
+ *Export as PDF*: Presentations can be exported to PDF.
+ *Export as HTML*: Presentations are deployed as HTML.
+ *Slide Numbering*: Supports slide numbering.

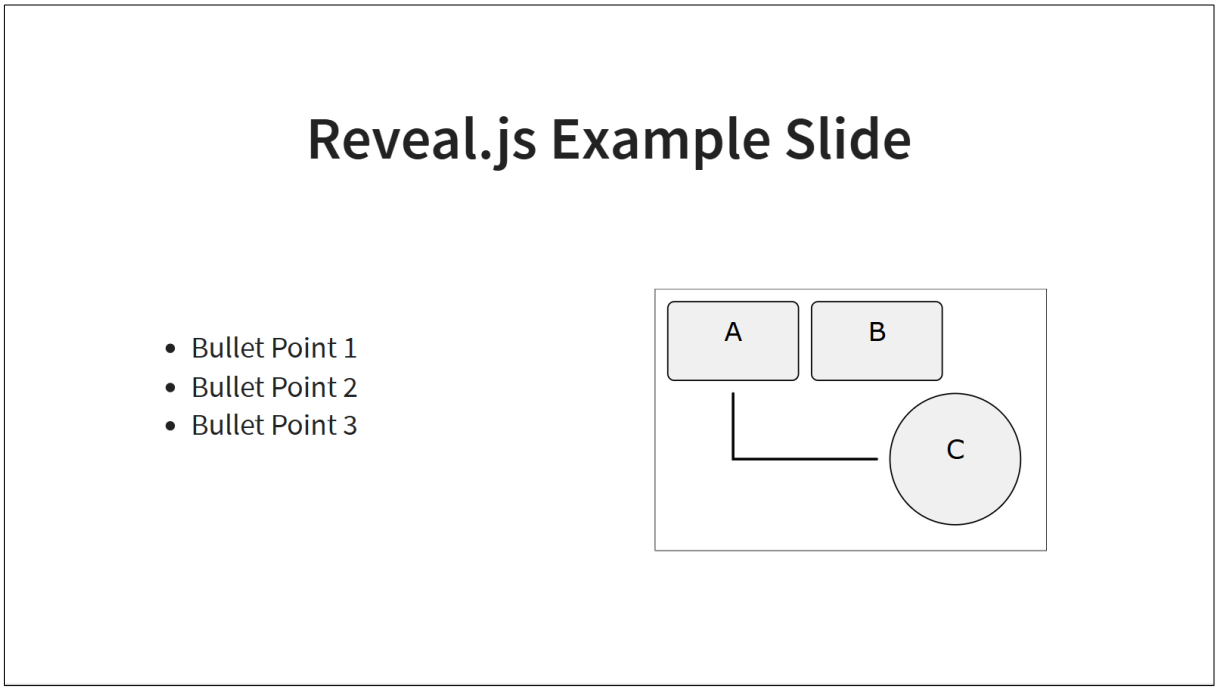


Figure 3.8: Reveal.js: Example slide. The corresponding source code is shown in Listing 3.8.

+ *Open Source*: Open-source project.
+ *Swipe Navigation*: Supports swipe navigation.
+ *Margin Navigation*: Supports margin navigation.
+ *Presenter Mode*: Presenter mode with speaker notes.

Limitations of Reveal.js include:

- *Collaboration*: No built-in support for collaboration.
- *Export as PowerPoint*: Not possible to export as PowerPoint.
- *Live Code Integration*: Does not offer live code integration.
- *Figure Zooming and Panning*: Lacks built-in functionality for figure zooming and panning.

3.3.4 impress.js [2011-]

impress.js is a presentation framework inspired by the idea behind Prezi [Halácsy et al. 2026]. It uses CSS3 transforms and transitions to create visually stunning and dynamic slide decks by positioning, rotating, and scaling them on an infinite canvas [Szopka and Ingo 2025]. This can be seen in Figure 3.9. To achieve this, each slide must include extra parameters for position, rotation, and scaling as HTML data attributes. The central position of a slide is defined by `data-x`, `data-y`, and `data-z`. Rotation is adjusted with `data-rotate-x`, `data-rotate-y`, and `data-rotate-z`. The scale of the slide is specified by `data-scale`, which defaults to 1. An example of a simple slide, where the slide is placed on the canvas can be seen in Figure 3.10, the corresponding source code is shown in Listing 3.9.

The key features of impress.js include:

+ *SVG Inclusion*: Supports SVG graphics through native HTML embedding.

```
<body>
<div class="reveal">
  <div class="slides">
    <section>
      <h2 class ="title">Reveal.js Example Slide</h2>
        <div class="columns two">
          <ul>
            <li>Bullet Point 1</li>
            <li>Bullet Point 2</li>
            <li>Bullet Point 3</li>
          </ul>
          <div class="image">
            <img src="diagrams/diagram.svg" alt="Reveal.js">
          </div>
        </div>
    </section>
  </div>
</div>

<script src="dist/reveal.js"></script>
<script src="plugin/notes/notes.js"></script>
<script>
Reveal.initialize({
    hash: true,
});
</script>
</body>
```

Listing 3.8: Reveal.js: Example slide source code. The slide deck is wrapped in `<div class="reveal">` and `<div class="slides">` elements. Each individual slide is defined by a `<section>` element. Calling the `Reveal.initialize()` function creates the finished slide deck. The resulting slide is shown in Figure 3.8.

+ *Globally Change Bullet Spacing*: Global bullet spacing can be adjusted in shared CSS styles.
+ *Globally Change Bullet Indentation*: Global bullet indentation can be adjusted in shared CSS styles.
+ *Export as HTML*: Presentations are deployed as HTML.
+ *Slide Numbering*: Supports slide numbering.
+ *Open Source*: Open-source project.
+ *Swipe Navigation*: Supports swipe navigation.
+ *Margin Navigation*: Supports margin navigation.
+ *Figure Zooming and Panning*: Provides zooming and panning as a core navigation mechanism.
+ *Presenter Mode*: Presenter mode support.

Limitations of impress.js include:

- *Collaboration*: No built-in support for collaboration.
- *Export as PDF*: Does not support exporting to PDF.

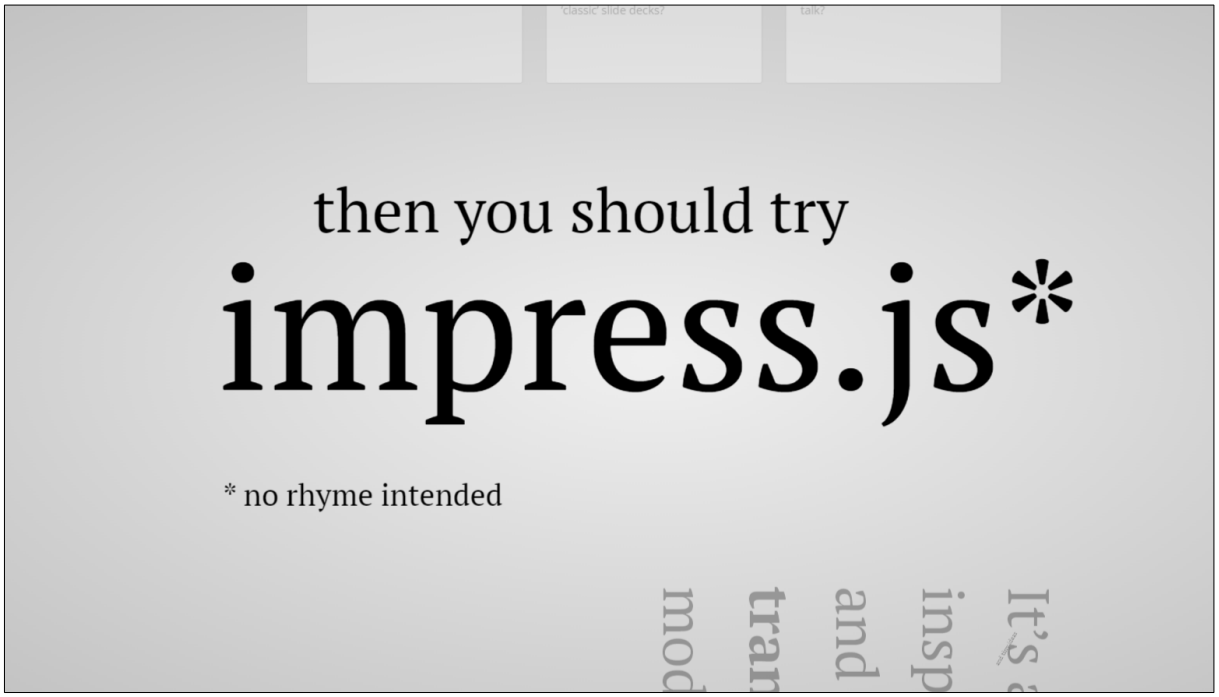


Figure 3.9: impress.js: All slides are placed on a canvas, which can then be traversed with a camera.

- *Export as PowerPoint*: Does not support exporting to PowerPoint.
- *Live Code Integration*: Does not offer live code integration.
- *High Resource Usage*: Intensive use of CSS3 transforms can be resource-heavy.

3.3.5 Bespoke.js [2014-2020]

Bespoke.js is a highly modular presentation library designed for building slide decks in modern web applications [Dalgleish 2020]. The core library sets up the presentation structure, provides an intuitive control API, and manages events. Its modular design allows for extensive customisation, as additional features, functionality, and themes can be implemented through a vast selection of plugins offered on NPM [NPM 2026b]. Bespoke.js can be locally installed with the NPM command:

```
npm install -g generator-bespoke
```

Bespoke.js uses the Gulp task runner to automate and enhance workflows [Bublitz and Schoffstall 2026]. When the project is created, the gulp command:

```
gulp serve
```

can be used to run a preview server with LiveReload. A simple example slide can be seen in Figure 3.11 and the corresponding source code in Listing 3.10.

The key features of Bespoke.js include:

+ *SVG Inclusion*: Supports SVG graphics through native HTML embedding.
+ *Globally Change Bullet Spacing*: Global bullet spacing can be adjusted in shared theme styles.
+ *Globally Change Bullet Indentation*: Global bullet indentation can be adjusted in shared theme styles.

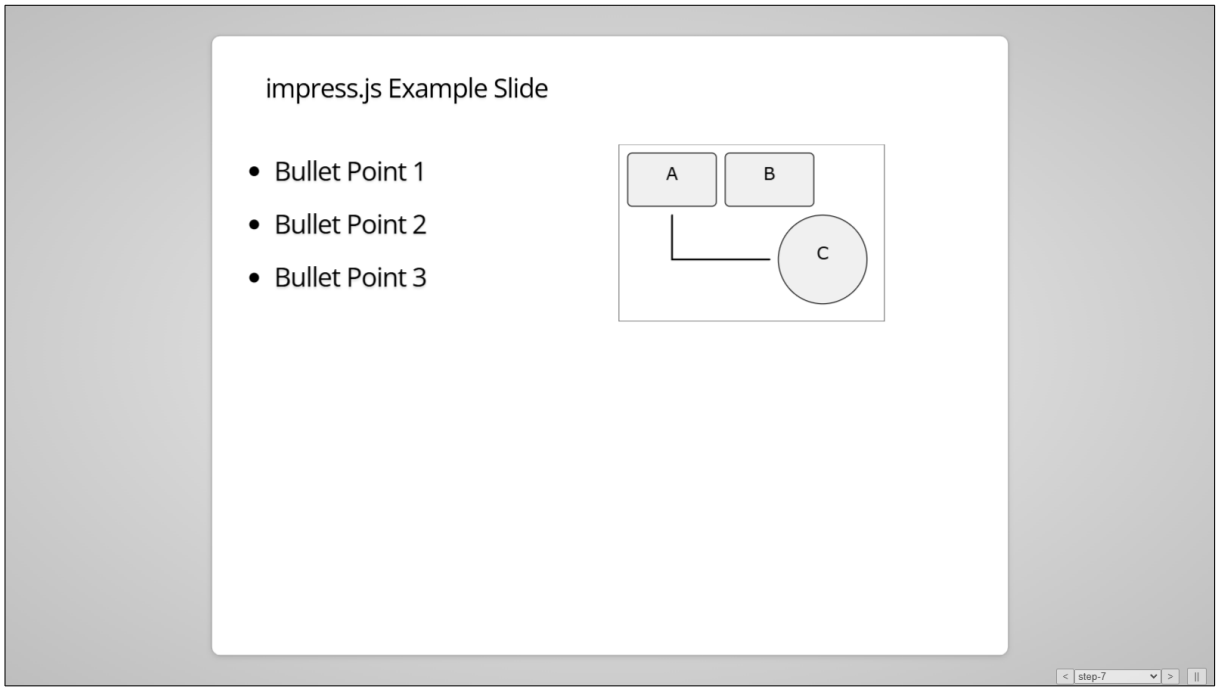


Figure 3.10: impress.js: Example slide. The corresponding source code is shown in Listing 3.9.

```
<div class="step slide" data-x="2000" data-y="-1500">
  <h1>impress.js Example Slide</h1>
  <br>
  <div class="container">
    <div class="column">
      <ul>
        <li>Bullet Point 1</li>
        <li>Bullet Point 2</li>
        <li>Bullet Point 3</li>
      </ul>
    </div>
    <div class="column">
      <img src="diagrams/diagram.svg" alt="Example image">
    </div>
  </div>
</div>
```

Listing 3.9: impress.js: Example slide source code. The resulting slide is positioned on the impress.js canvas with the `data-x` and `data-y` attributes, as shown in Figure 3.10.

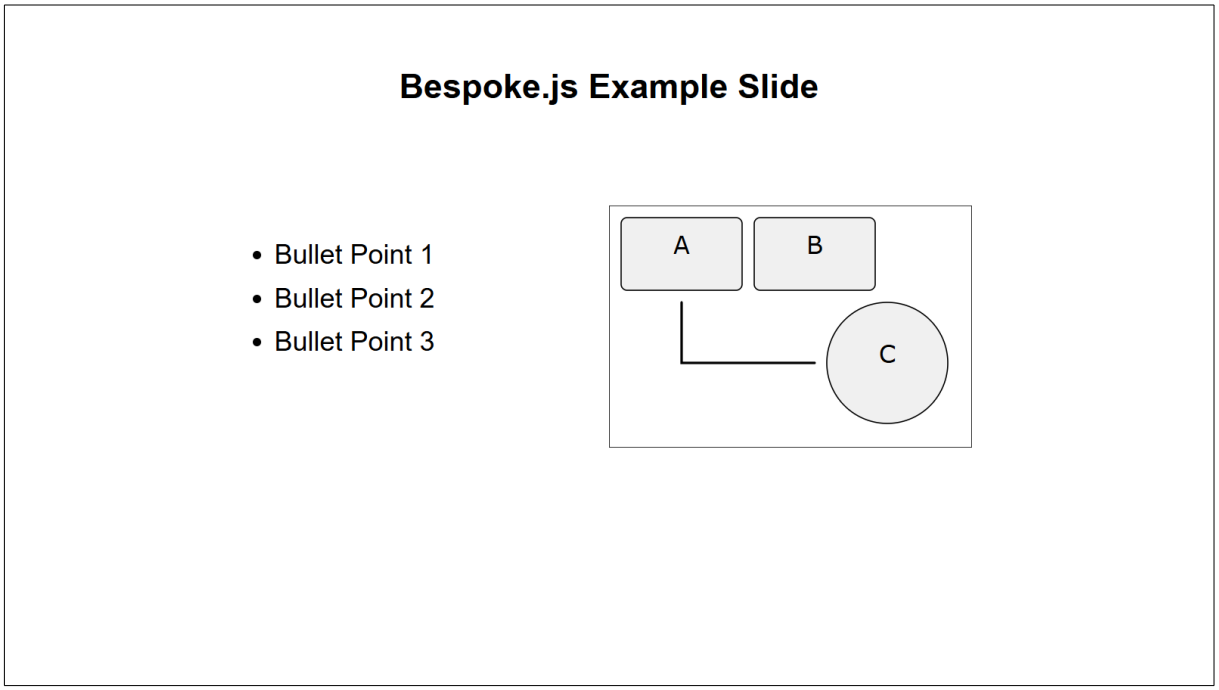


Figure 3.11: Bespoke.js: Example slide. The corresponding source code is shown in Listing 3.10.

+ *Export as PDF*: Presentations can be exported to PDF.
+ *Export as HTML*: Presentations are deployed as HTML.
+ *Slide Numbering*: Supports slide numbering.
+ *Open Source*: Open-source project.
+ *Swipe Navigation*: Supports swipe navigation.
+ *Margin Navigation*: Supports margin navigation.
+ *Figure Zooming and Panning*: Zoom and pan support through plugins.
+ *Presenter Mode*: Presenter mode support.

Limitations of Bespoke.js include:

- *Collaboration*: No built-in support for collaboration.
- *Export as PowerPoint*: Not possible to export as PowerPoint.
- *Live Code Integration*: Does not offer live code integration.
- *Limited Native Features*: As a modular framework, Bespoke.js does not include many advanced features natively.

3.3.6 Inspire.js [2018-]

Inspire.js, formerly knows as CSS-based SlideShow System (CSSS), is a JavaScript-based slide tool with a plugin-based architecture, allowing users to easily extend its functionality [Verou 2026]. Slides are defined by adding the `slide` class to any block-level element, as shown in Figure 3.12 and Listing 3.11.

The key features of Inspire.js include:

```
<section>
  <h2>Bespoke.js Example Slide</h2>
  <div class="slide-content">
   <div class="bullets">
    <ul>
      <li>Bullet Point 1</li>
      <li>Bullet Point 2</li>
      <li>Bullet Point 3</li>
    </ul>
   </div>
   <div class="image">
     <img src="diagrams/diagram.svg" alt="Inspire.js">
   </div>
  </div>
</section>

<style>
.slide-content {
  display: flex;
    justify-content: space-between;
    padding-top: 2rem;
    width: 80%;
  }
  .bullets {
    width: 50%;
    display: flex;
    align-items: flex-start;
    padding-top:  1rem;
    padding-left: 0;
  }
  .image {
    width: 50%;
    display: flex;
    align-items: center;
    justify-content: center;
  }
</style>
```

Listing 3.10: Bespoke.js: Example slide source code. Slide content is placed inside a `<section>` element. The resulting slide is shown in Figure 3.11.

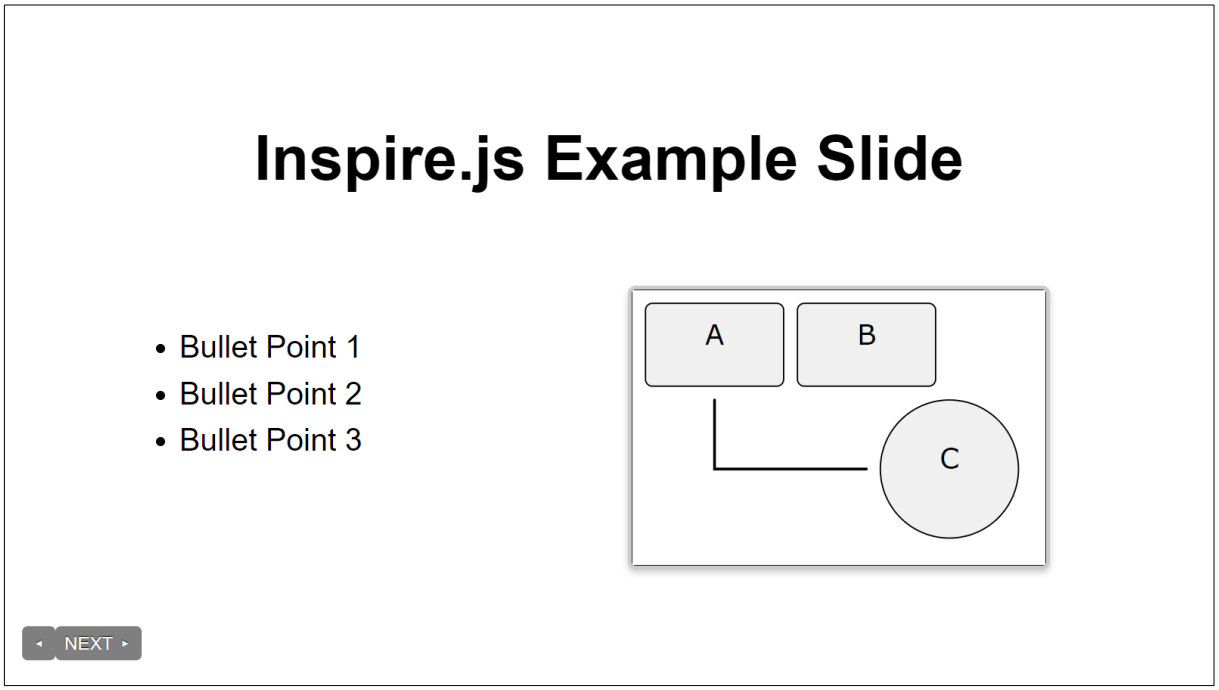


Figure 3.12: Inspire.js: Example slide. The corresponding source code is shown in Listing 3.11.

+ *SVG Inclusion*: Supports SVG graphics through native HTML embedding.
+ *Globally Change Bullet Spacing*: Global bullet spacing can be adjusted in shared CSS styles.
+ *Globally Change Bullet Indentation*: Global bullet indentation can be adjusted in shared CSS styles.
+ *Export as HTML*: Presentations are deployed as HTML.
+ *Slide Numbering*: Supports slide numbering.
+ *Open Source*: Open-source project.
+ *Swipe Navigation*: Supports swipe navigation.
+ *Margin Navigation*: Supports margin navigation.
+ *Presenter Mode*: Presenter mode through plugins.

Limitations of Inspire.js include:

- *Collaboration*: No built-in support for collaboration.
- *Export as PDF*: Does not support exporting to PDF.
- *Export as PowerPoint*: Does not support exporting to PowerPoint.
- *Live Code Integration*: Does not offer live code integration.
- *Figure Zooming and Panning*: Lacks built-in functionality for figure zooming and panning.

```
<article class="slide" id="example-slide">
<h1>Inspire.js Example Slide</h1>
<div class="slide-content">
  <div class="bullets">
    <ul>
      <li>Bullet Point 1</li>
      <li>Bullet Point 2</li>
      <li>Bullet Point 3</li>
    </ul>
  </div>
  <div class="image">
    <img src="./diagrams/diagram.svg" alt="Inspire.js">
  </div>
</div>
</article>
```

Listing 3.11: Inspire.js: Example slide source code. The slide content is placed inside an `<article class="slide">` element. The resulting slide is shown in Figure 3.12.

3.3.7 Comparison of JavaScript-Based Slide Decks

JavaScript-based slide deck tools are compared in Table 3.3. All of the JavaScript-based slide deck tools support the native input of SVG graphics and slide numbering, but they all lack standalone collaborative editing capabilities. Each tool offers customisation options for bullet indentation and spacing in master templates. Exporting presentations as PDFs is possible with Bespoke.js, Reveal.js, and Shower. Furthermore, since all of these tools are built on HTML, they all support exporting content in that format. However, none of them provide the option to directly export to PowerPoint, which could be a limitation for users needing that specific format.

The update frequency for these slide deck tools varies, with Inspire.js and Reveal.js having recent updates, while Deck.js has not been updated since 2016. In terms of popularity, Reveal.js stands out with 67.4k GitHub stars, followed by impress.js with 37.6k GitHub stars, indicating their widespread use and an active community.

Navigation features such as margin tap and swipe gestures to navigate slides are supported by all the tools. A presenter mode, which provides a second window for speaker notes, is available across all slide deck tools. Additionally, all the slide deck tools are distributed under the MIT license and are open-source.

3.4 Hosted Slide Decks

Hosted slide deck tools are online tools, which allow users to create, manage, and share presentation slides over the internet without needing to install software locally. These tools offer a range of modern features such as AI-powered tools and built-in, drag-and-drop editors. Presentations can be shared via a simple link, allowing for easy access and real-time collaboration without downloading files. Many tools also offer embedding options for web sites and built-in integrations with productivity suites such as Microsoft 365 and Google Workspace. Most hosted slide deck tools have free versions with limited features, but offer subscription models for different advanced features such as AI-designs and PDF export. This section describes six popular hosted tools, which can be used to create online presentations: Google Slides, Zoho Show, Prezi, Visme, Mentimeter, and Microsoft Sway.

Feature	Shower	Reveal.js	Deck.js	impress.js	Bespoke.js	Inspire.js
Collaboration:	✗	✗	✗	✗	✗	✗
SVG Inclusion:	✓	✓	✓	✓	✓	✓
Globally Change Bullet Spacing:	✓	✓	✓	✓	✓	✓
Globally Change Bullet Indentation:	✓	✓	✓	✓	✓	✓
Export as PDF:	✓	✓	✗	✗	✓	✗
Export as HTML:	✓	✓	✓	✓	✓	✓
Export as PowerPoint:	✗	✗	✗	✗	✗	✗
Slide Numbering:	✓	✓	✓	✓	✓	✓
Live Code Integration:	✗	✗	✗	✗	✗	✗
Open Source:	✓	✓	✓	✓	✓	✓
Licence:	MIT	MIT	MIT	MIT	MIT	MIT
First Release:	2010-10-25	2011-06-07	2011-06-24	2011-12-28	2012-12-13	2018-09
Last Update:	2024-11-20	2025-12-23	2016-05-04	2025-09-25	2020-09-08	2025-11-18
Popularity (GitHub stars):	4.8k	70.4k	5.4k	38.6k	4.7k	1.7k
Swipe Navigation:	✓	✓	✓	✓	✓	✓
Margin Navigation:	✓	✓	✓	✓	✓	✓
Figure Zooming and Panning:	✗	✗	✗	✓	✓	✗
Presenter Mode:	✓	✓	✓	✓	✓	✓

Table 3.3: Comparison of JavaScript-based slide deck tools.

3.4.1 Google Slides [2006-]

Google Slides is part of the free, web-based Google Docs suite offered by Google [Google 2026b]. It allows users to collaborate on presentations in real-time, and present slide decks online. Google Slides can easily be exported as a PowerPoint presentation or PDF. A simple Google Slides example slide can be seen in Figure 3.13.

The key features of Google Slides include:

+ *Collaboration*: Supports real-time collaborative editing.
+ *Globally Change Bullet Indentation*: Bullet indentation can be adjusted through master slides.
+ *Export as PDF*: Supports exporting as PDF.
+ *Export as PowerPoint*: Supports exporting as PowerPoint.
+ *Slide Numbering*: Supports slide numbering.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Margin Navigation*: Supports margin navigation.
+ *Presenter Mode*: Presenter mode with speaker notes.

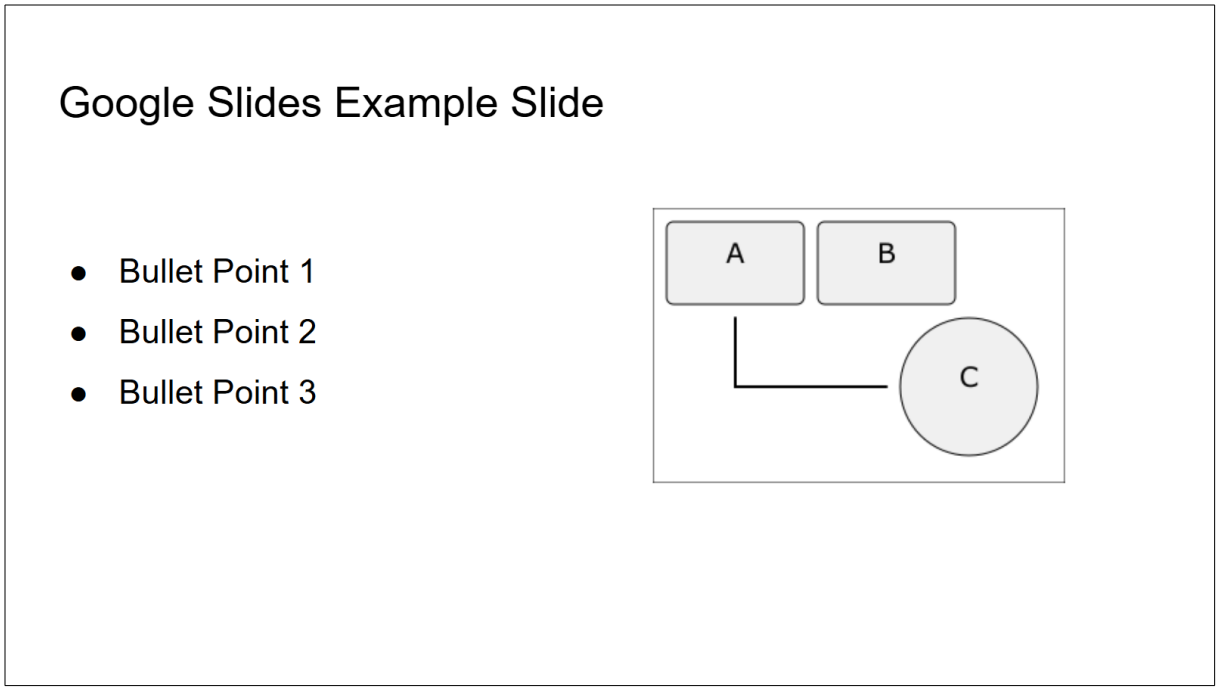


Figure 3.13: Google Slides: Example slide with an PNG image on the right (Google Slides does not support SVG).

Limitations of Google Slides include:

- *SVG Inclusion*: Not possible to include SVG vector graphics.
- *Globally Change Bullet Spacing*: Not possible to change bullet spacing in master slides.
- *Export as HTML*: Does not support exporting as standalone HTML.
- *Live Code Integration*: Does not support integration of live code.
- *Open Source*: Proprietary system.
- *Figure Zooming and Panning*: Does not provide built-in zoom or pan features.

3.4.2 Zoho Show [2006-]

Zoho Show is a collaborative presentation tool for modern teams [Zoho 2026]. It offers a user-friendly platform for creating, editing, and delivering professional presentations. Zoho Show is mostly known for its wide range of templates, themes, and design tools, as can be seen in Figure 3.14. Users can easily customise these themes and templates to their needs. As part of the Zoho suite, Zoho Show integrates seamlessly with other Zoho applications. It offers many features and it shares similarities with Visme, like the very user-friendly drag-and-drop editor. Due to its many features and huge range of templates and themes, it can be overwhelming and hard to navigate. Certain features, such as offline mode, advanced real-time collaboration, enhanced file sharing, custom branding, and additional integrations, are only available with paid plans. A simple example slide can be seen in Figure 3.15.

The key features of Zoho Show include:

+ *Collaboration*: Supports real-time collaboration.
+ *Globally Change Bullet Spacing*: Bullet spacing can be adjusted through master templates.

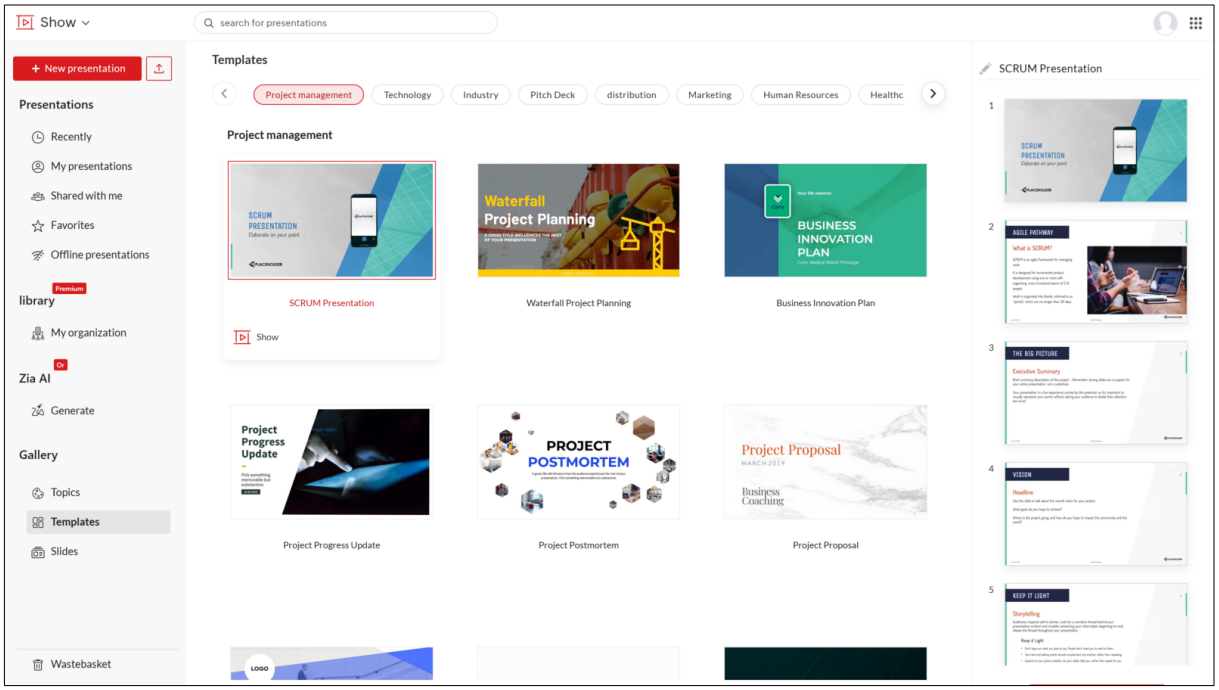


Figure 3.14: Zoho Show: Templates with almost fully finished slides to build up on.

+ *Globally Change Bullet Indentation*: Bullet indentation can be adjusted through master templates.
+ *Export as PDF*: Supports exporting as PDF.
+ *Export as HTML*: Supports exporting as HTML.
+ *Export as PowerPoint*: Supports exporting as PowerPoint.
+ *Slide Numbering*: Supports slide numbering.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Margin Navigation*: Supports margin navigation.
+ *Presenter Mode*: Presenter mode with speaker notes.

Limitations of Zoho Show include:

- *SVG Inclusion*: Does not support the inclusion of vector graphics (SVG).
- *Live Code Integration*: Does not support the integration of live code within presentations.
- *Open Source*: Proprietary system.
- *Figure Zooming and Panning*: Does not provide built-in zoom or pan features for presentation elements.
- *Advanced Features*: Many advanced features and templates are restricted to paid users.

3.4.3 Prezi [2009-]

Prezi is an engaging web-based presentation tool that focuses on animations and offers a map-like overview allowing users to navigate between topics and slides, and zoom in and out for slide details [Halácsy et al.

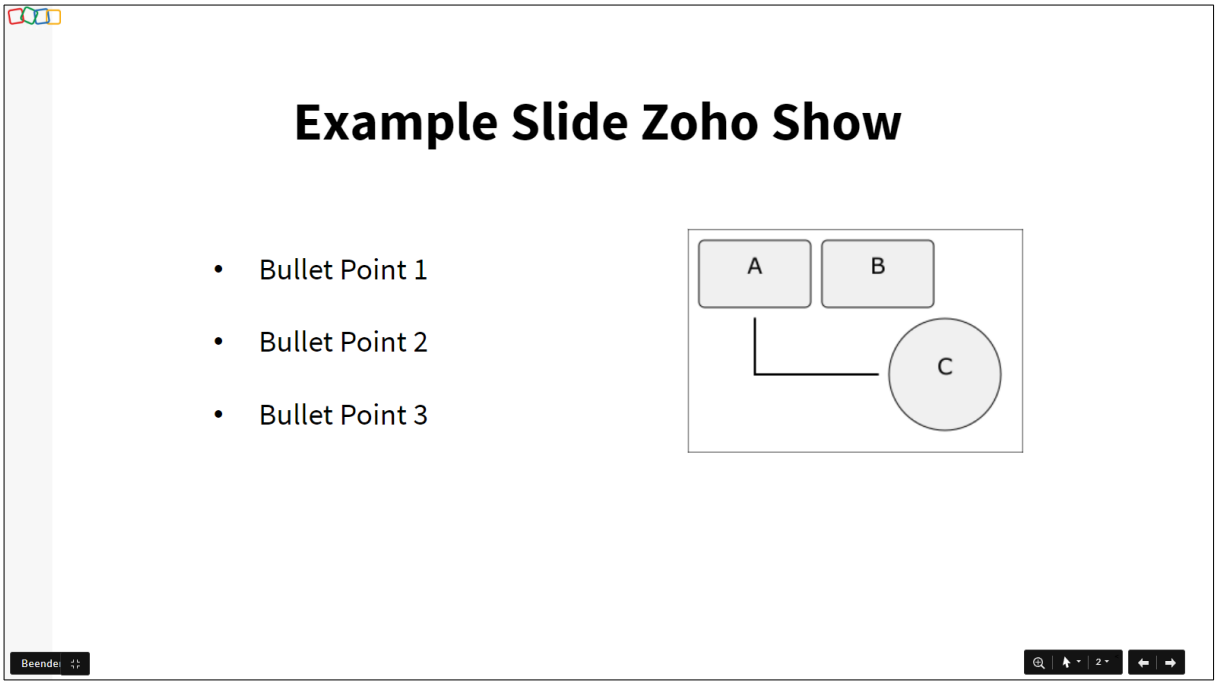


Figure 3.15: Zoho Show: Example slide with an PNG image on the right. Zoho Show does not support import of SVG vector graphics.

2026]. With Prezi, users can create visually stunning slide decks, where the slides are placed on a very large canvas, similar to impress.js. An example slide can be seen in Figure 3.16. The corresponding canvas the slide is placed on is shown in Figure 3.17. Prezi offers a tiered pricing model with various plans designed for different user needs. The plans include a free basic version with limited features, and several paid subscriptions, such as individual, business, and education.

The key features of Prezi include:

+ *Collaboration*: Real-time collaboration, depending on the selected plan.
+ *Export as PDF*: Supports exporting as PDF.
+ *Slide Numbering*: Supports slide numbering.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Figure Zooming and Panning*: Core zoom and pan interaction on a presentation canvas.
+ *Presenter Mode*: Presenter mode support.

Limitations of Prezi include:

- *SVG Inclusion*: Not possible to include SVG vector graphics.
- *Globally Change Bullet Spacing*: Limited global control of bullet spacing.
- *Globally Change Bullet Indentation*: Limited global control of bullet indentation.
- *Export as HTML*: Not possible to export as standalone HTML.
- *Export as PowerPoint*: Not possible to export as PowerPoint.
- *Live Code Integration*: Does not support integration of live code.

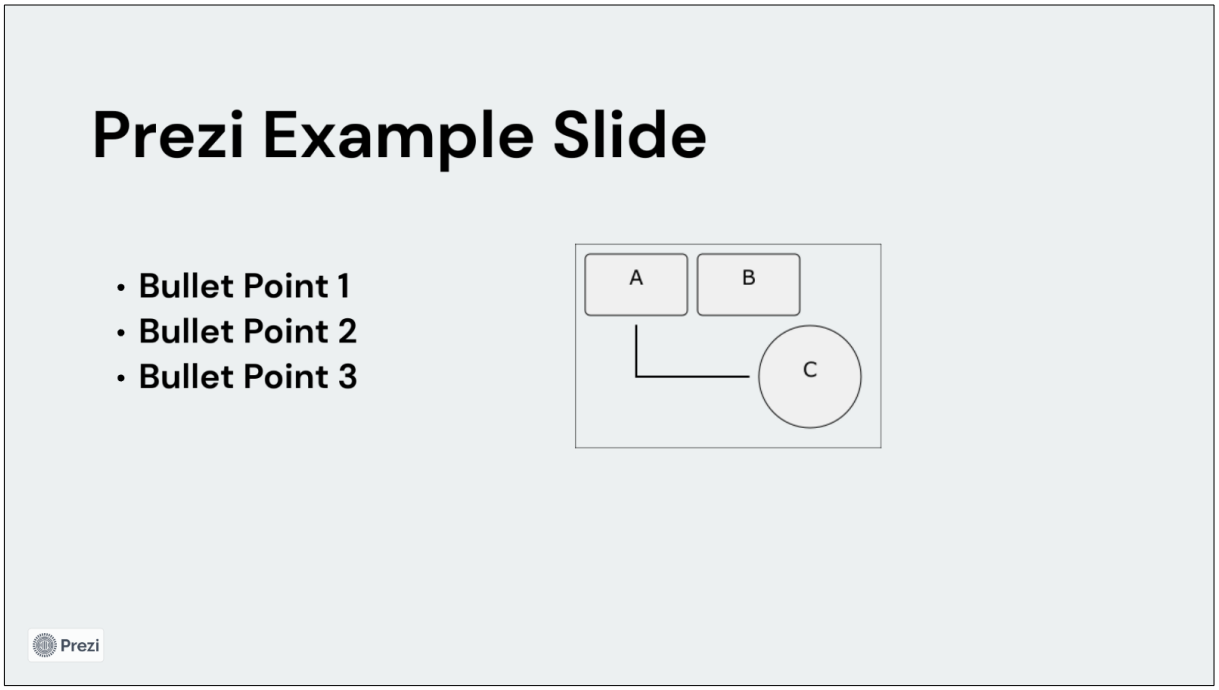


Figure 3.16: Prezi: Example slide. The slide is placed on a canvas. The colour palette of the slide and canvas can only be fully customised with a paid subscription.

- *Open Source*: Proprietary system.
- *Margin Navigation*: Does not support margin tapping for navigation in the viewer.

3.4.4 Visme [2013-]

Visme is an all-in-one visual content creation tool that allows users to design a wide range of material, such as presentations, infographics, reports, social media graphics, and more [Taei 2026a]. It simplifies content creation with its user-friendly interface, while offering a modern drag-and-drop tool, as shown in Figure 3.18. Although Visme offers many tools, most of the features are only available with a paid subscription, such as real-time collaboration, expanded download options, advanced customisation tools, brand kits, analytics, and offline presentation [Taei 2026b]. An example slide can be seen in Figure 3.19

The key features of Visme include:

+ *Collaboration*: Supports real-time collaboration.
+ *SVG Inclusion*: Possible to include SVG vector graphics.
+ *Export as PDF*: Supports exporting as PDF.
+ *Export as HTML*: Supports exporting as HTML.
+ *Export as PowerPoint*: Supports exporting as PowerPoint.
+ *Slide Numbering*: Supports slide numbering.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Presenter Mode*: Presenter mode support.

Limitations of Visme include:

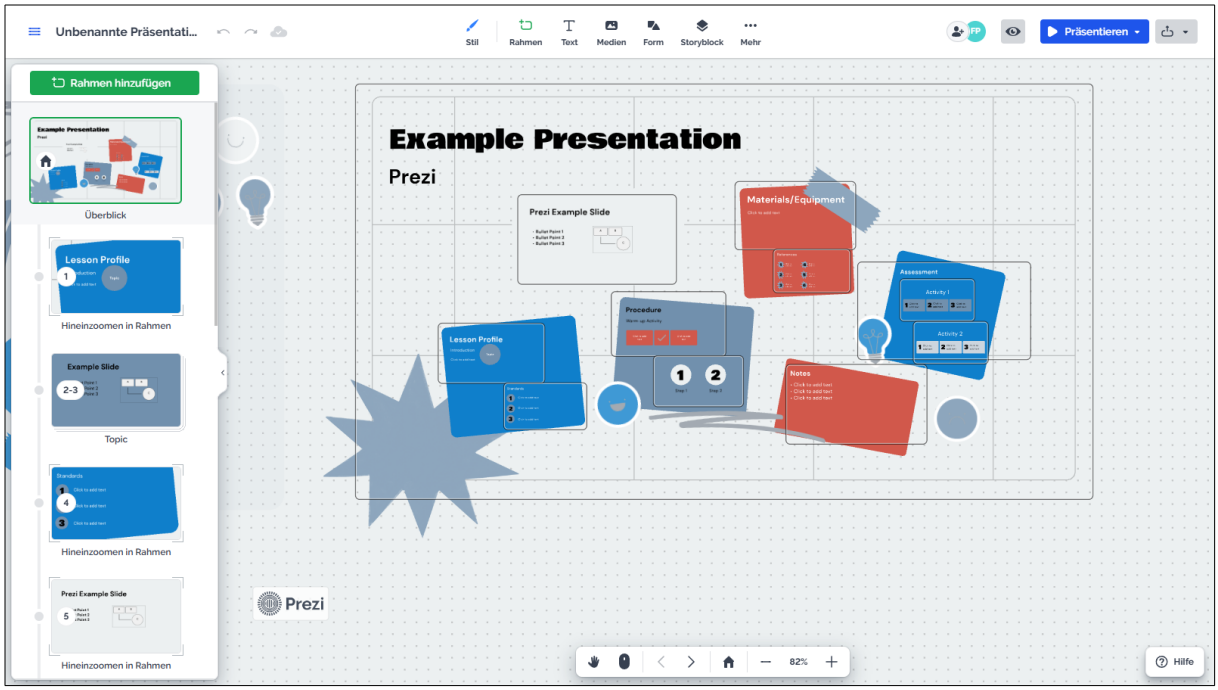


Figure 3.17: Prezi: Canvas with presentation slides. When presenting, the author can zoom in and out of the canvas to reach specific slides.

- *Globally Change Bullet Spacing*: Limited global control of bullet spacing in master templates.
- *Globally Change Bullet Indentation*: Limited global control of bullet indentation in master templates.
- *Live Code Integration*: Does not support integration of live code.
- *Open Source*: Proprietary system.
- *Margin Navigation*: Does not support margin tapping for navigation in the viewer.
- *Figure Zooming and Panning*: Does not provide built-in zoom or pan features.
- *Advanced Features*: Many advanced features and templates are restricted to paid users.

3.4.5 Mentimeter [2014-]

Mentimeter is an interactive presentation tool, which allows presenters to engage their audience in real-time [Warström et al. 2026]. Unlike most other slide deck tools, in Mentimeter, users can create various types of interactive content such as polls, quizzes, word clouds, and Q&A sessions. Audience members can participate using their smartphones or other devices by accessing a unique code or link provided by the presenter. Mentimeter provides visualisations of the responses in real-time, making presentations more dynamic and engaging. However, some of its advanced features such as unlimited slides, advanced audience interaction tools (such as moderation of Q&A), customised branding options and the ability to export data to Excel, are only available with paid plans. It is often used in classrooms, meetings, workshops, and conferences to gather feedback and enhance audience involvement, mostly with the help of polls. A simple example slide can be seen in Figure 3.20. To illustrate one of Mentimeter's most used features, a test poll was created, as shown in Figure 3.21.

The key features of Mentimeter include:

+ *Collaboration*: Supports real-time collaborative editing.

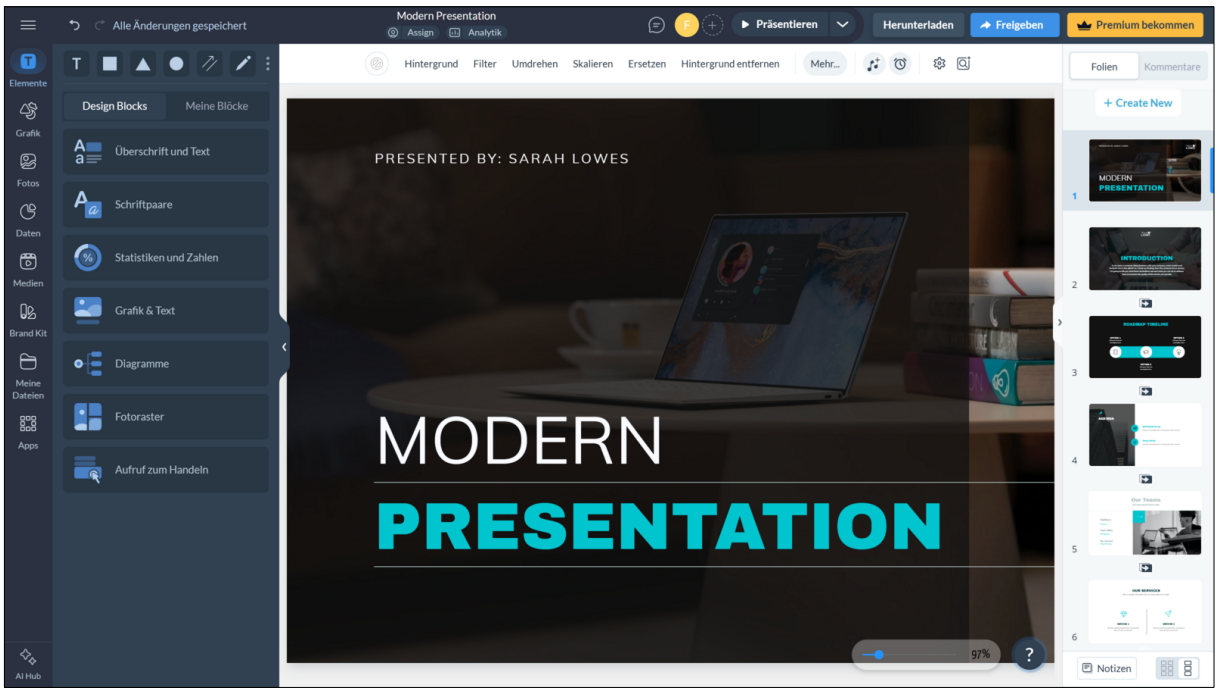


Figure 3.18: Visme: User friendly drag-and-drop editor.

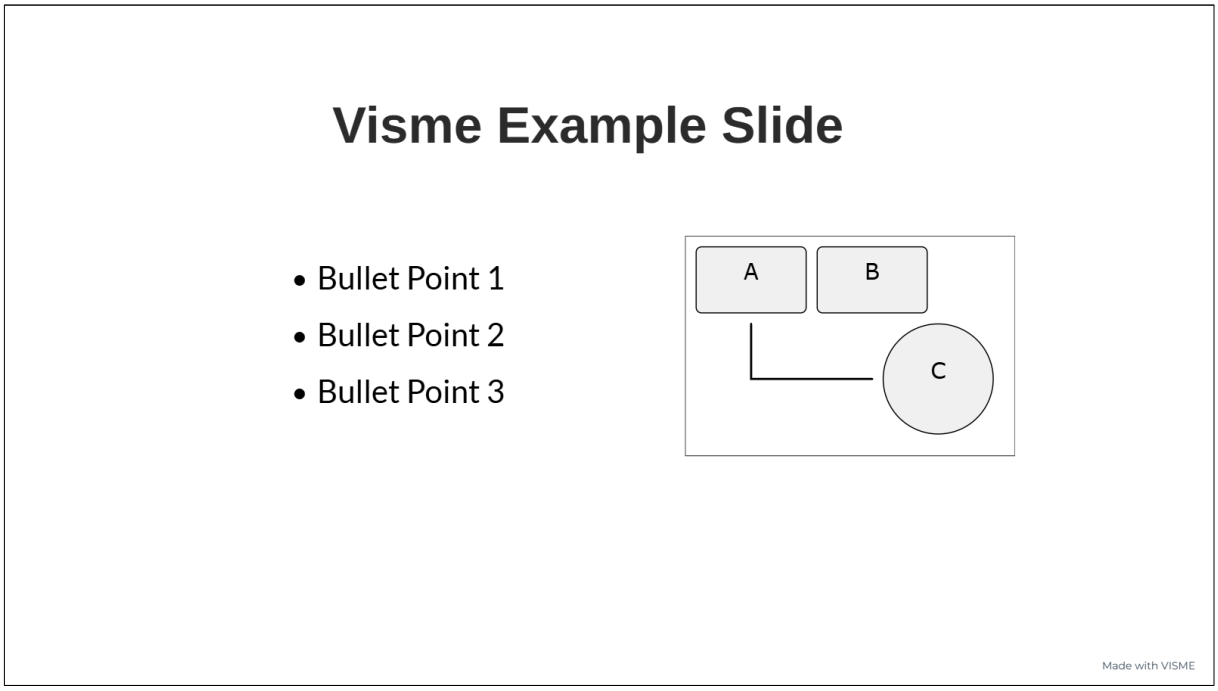


Figure 3.19: Visme: Example slide in its predefined presentation window. The slide can only be fully seen or downloaded with a paid subscription model.

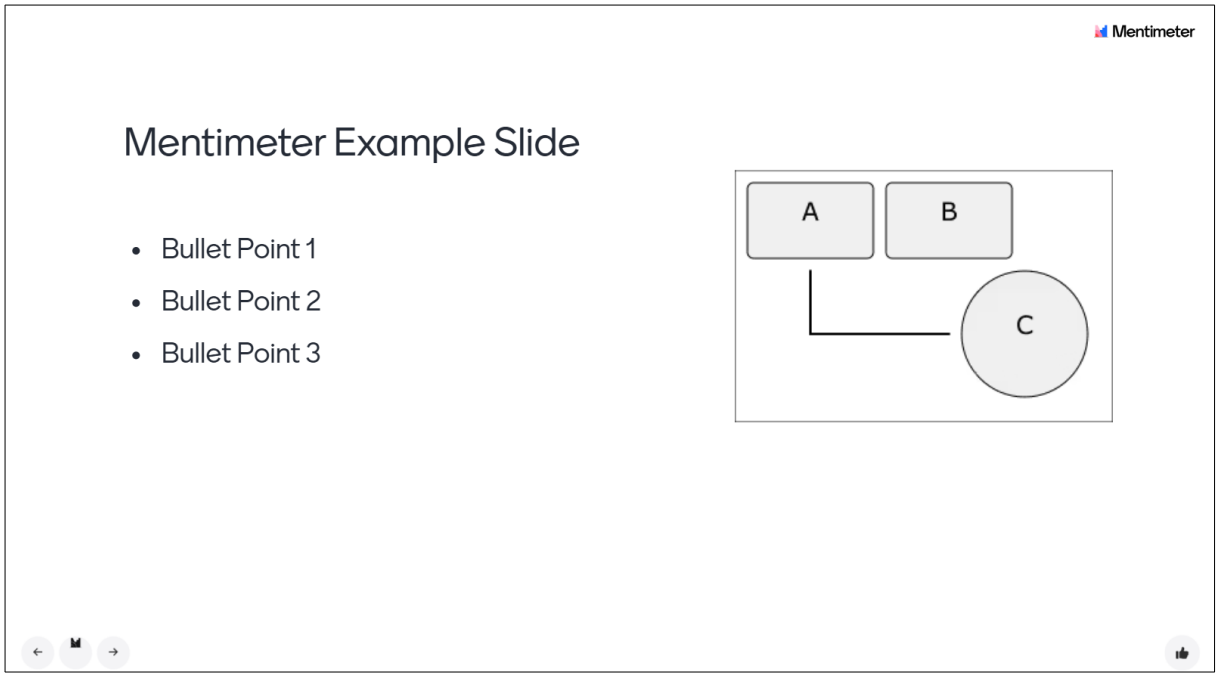


Figure 3.20: Mentimeter: Example slide with an SVG vector graphic on the right.

+ *SVG Inclusion*: Supports SVG vector graphics.
+ *Export as PDF*: Supports exporting as PDF.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Presenter Mode*: Presenter mode support.

Limitations of Mentimeter include:

- *Globally Change Bullet Spacing*: No global bullet spacing control.
- *Globally Change Bullet Indentation*: No global bullet indentation control.
- *Export as HTML*: Not possible to export as standalone HTML.
- *Export as PowerPoint*: Not possible to export as PowerPoint.
- *Slide Numbering*: Does not support slide numbering.
- *Live Code Integration*: Does not support integration of live code.
- *Open Source*: Proprietary system.
- *Margin Navigation*: Does not support margin tapping for navigation.
- *Figure Zooming and Panning*: Does not provide built-in zoom or pan features.
- *Advanced Features*: Many advanced features are restricted to paid plans.

3.4.6 Microsoft Sway [2015-]

Microsoft Sway is an app from Microsoft Office that makes it easy to create and share interactive reports, personal stories, presentations, and more [Microsoft 2026b]. Content in Microsoft Sway is organised

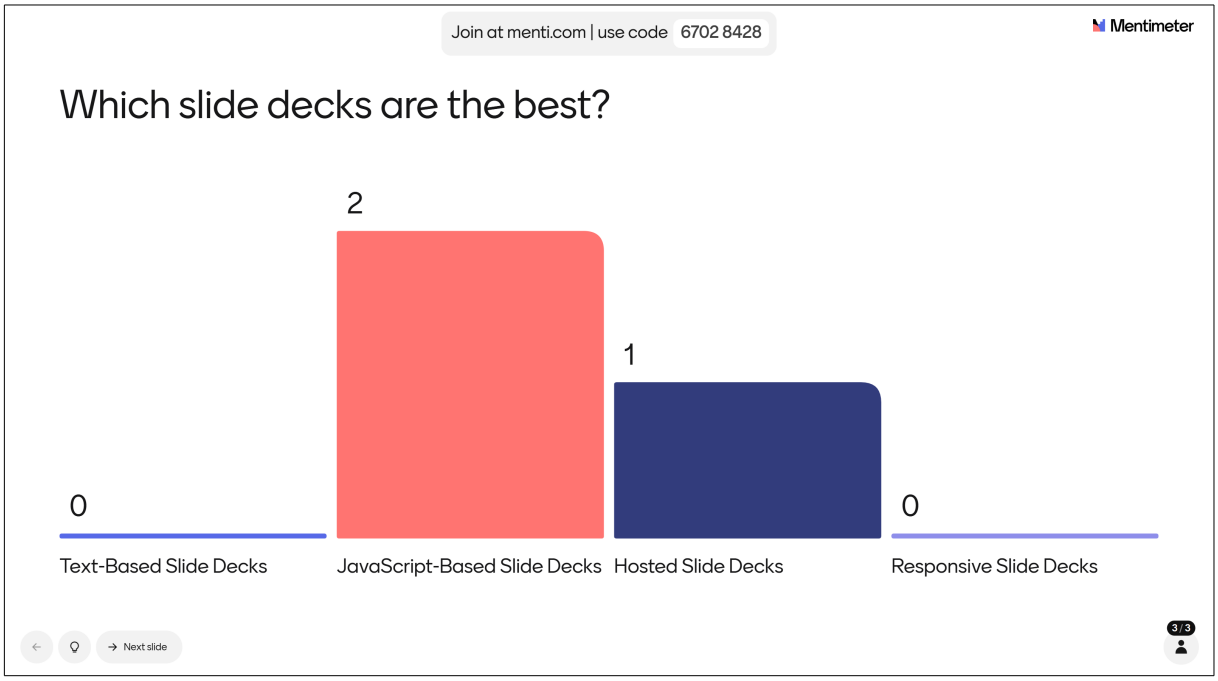


Figure 3.21: Mentimeter: Example poll about which slide decks are the best, based on an example user voting.

as a storyline, where users can add text, images and multimedia elements. Its built-in design engine automatically formats the content. Users can switch styles and designs, but are limited in terms of options. Microsoft Sway automatically optimises presentations for different devices, ensuring a refined look without the need for design expertise. As part of the Microsoft Office 365, Microsoft Sway can be easily integrated in other Microsoft services. An example slide with the standard pre-defined Microsoft Sway design can be seen in Figure 3.22.

The key features of Microsoft Sway include:

+ *Collaboration*: Supports real-time collaborative editing.
+ *Export as PDF*: Supports exporting as PDF.
+ *Swipe Navigation*: Supports swipe gestures.
+ *Figure Zooming and Panning*: Includes zoom and pan features for embedded images.

Limitations of Microsoft Sway include:

- *SVG Inclusion*: Does not support SVG inclusion.
- *Globally Change Bullet Spacing*: No global bullet spacing control.
- *Globally Change Bullet Indentation*: No global bullet indentation control.
- *Export as HTML*: Does not support exporting as standalone HTML.
- *Export as PowerPoint*: Does not support exporting as PowerPoint.
- *Slide Numbering*: Does not support slide numbering.
- *Live Code Integration*: Does not support integration of live code.

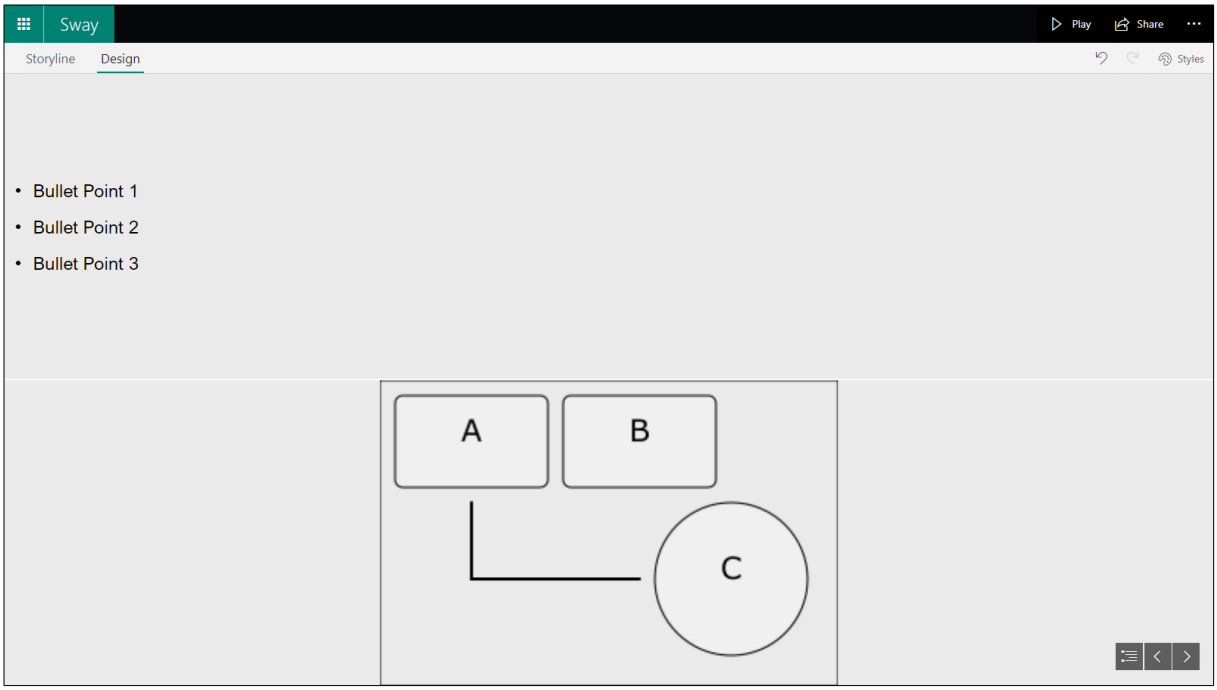


Figure 3.22: Microsoft Sway: Example slide with PNG image. Microsoft Sway does not support import of SVG vector graphics. The styling options in Microsoft Sway are intentionally limited, ensuring ease of use, cross-device compatibility, and streamlined visual storytelling.

- *Open Source*: Proprietary system.
- *Margin Navigation*: Does not support margin tapping for navigation.
- *Presenter Mode*: Does not provide a dedicated presenter mode.

3.4.7 Comparison of Hosted Slide Decks

While many hosted slide deck tools offer similar core features, there are notable differences around specific functionality, as summarised in Table 3.4. All tools support real-time collaboration, allowing multiple users to work on presentations simultaneously. However, only Visme and Mentimeter support the native import of vector graphics (SVG), making them ideal for design-heavy presentations.

Google Slides and Zoho Show allow changes to bullet indentation within the master template, but Visme, Prezi, Microsoft Sway, and Mentimeter do not support bullet indentation or spacing adjustments globally. When it comes to export options, all tools support PDF export, but only Visme, and Zoho Show offer the ability to export as HTML. PowerPoint export is supported by Google Slides, Visme and Zoho Show. Slide numbering is available across most tools, except for Microsoft Sway and Mentimeter, which do not follow traditional slide structures. While Prezi and Microsoft Sway offer zoom and pan features for interactive presentations, other tools lack this functionality.

All tools support swipe gestures for moving between slides. However, only Google Slides and Zoho Show also offer margin tapping for navigation. Google Slides, Prezi, Visme, Zoho Show, and Mentimeter support a dedicated presenter mode. Lastly, while all the tools are proprietary and are updated frequently, their popularity varies. Google Slides and Prezi are the most widely used, while Microsoft Sway, Visme, Zoho Show, and Mentimeter have a more moderate user base.

Feature	Google Slides	Zoho Show	Prezi	Visme	Mentimeter	Microsoft Sway
Collaboration:	✓	✓	✓	✓	✓	✓
SVG Inclusion:	✗	✗	✗	✓	✓	✗
Globally Change Bullet Spacing:	✗	✓	✗	✗	✗	✗
Globally Change Bullet Indentation:	✓	✓	✗	✗	✗	✗
Export as PDF:	✓	✓	✓	✓	✓	✓
Export as HTML:	✗	✓	✗	✓	✗	✗
Export as PowerPoint:	✓	✓	✗	✓	✗	✗
Slide Numbering:	✓	✓	✓	✓	✗	✗
Live Code Integration:	✗	✗	✗	✗	✗	✗
Open Source:	✗	✗	✗	✗	✗	✗
Licence:	Proprietary	Proprietary	Proprietary	Proprietary	Proprietary	Proprietary
First Release:	2006-03-09	2006	2009-04-05	2013	2014	2014-08
Last Update:	Frequent	Frequent	Frequent	Frequent	Frequent	Frequent
Popularity (Usage):	High	Medium	High	Medium	Medium	Medium
Swipe Navigation:	✓	✓	✓	✓	✓	✓
Margin Navigation:	✓	✓	✗	✗	✗	✗
Figure Zooming and Panning:	✗	✗	✓	✗	✗	✓
Presenter Mode:	✓	✓	✓	✓	✓	✗

Table 3.4: Comparison of hosted slide deck tools.

3.5 Responsive Slide Decks

Responsive web design is a modern approach to designing web sites and web applications which adapt to the characteristics of the end user's device. The same content is served to every device from the same URL, but its internal logic supports different layouts and input modalities. Responsive slide decks are web-based slide decks which can adapt themselves to various display devices, by responding to different screen widths, and supporting mouse, touch, and keyboard input.

3.5.1 Rslidy [2019-]

Rslidy is a modern, lightweight, and responsive presentation tool designed to run directly in web browsers using Living HTML, Modern CSS, and TypeScript [Rslidy 2026; Hipp 2021b]. Building on the idea of Slidy2 [Raggett 2006], Rslidy improves the original concept by embracing the technqiues of responsive web design to ensure that slide decks are viewable on a wide variety of end devices. Rslidy is much more sophisticated in terms of its feature set, with CSS variables for custom settings, a variety of template styles, built-in support for responsive tables, and extensive print settings. Furthermore, Rslidy is completely self-contained, requiring no additional dependencies. Slides are created within `<section>` or `<div class="slide">` elements, and are presented one at a time with transitions. An example slide can be seen in Figure 3.23 and the corresponding Listing 3.12.

The key features of Rslidy include:

+ *Standard HTML Slides*: Slides can be created with standard HTML elements.

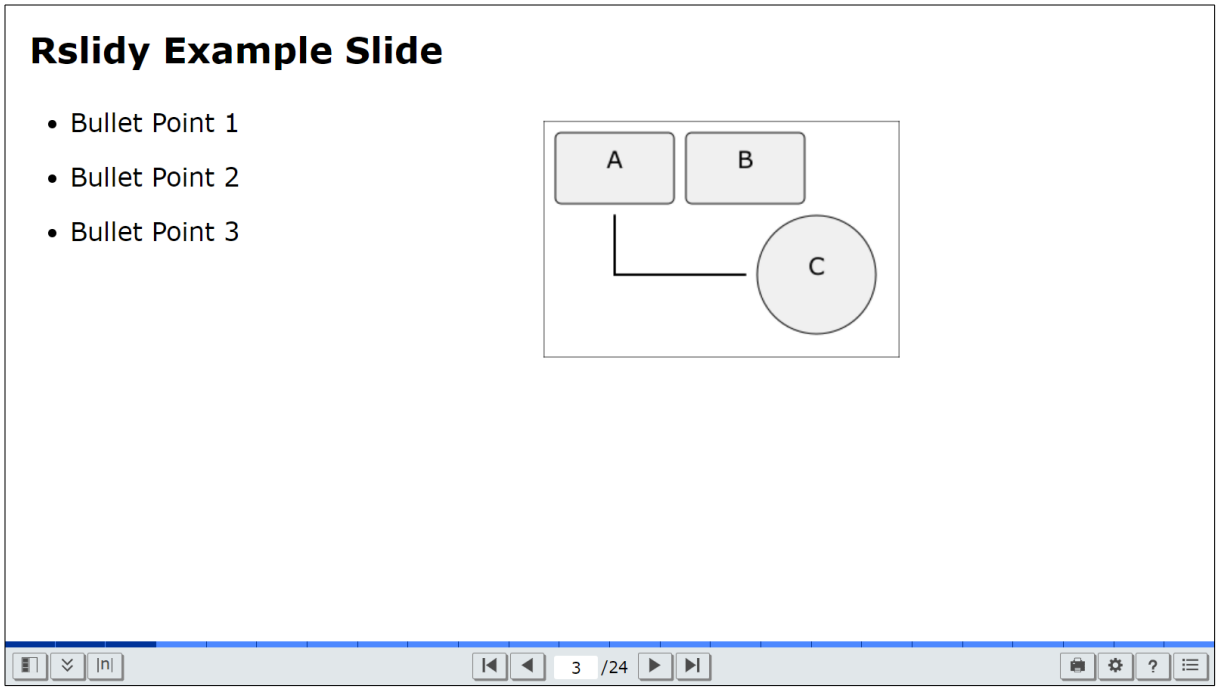


Figure 3.23: Rslidy: Example slide. The corresponding source code is shown in Listing 3.12.

+ *SVG Inclusion*: SVG vector graphics can be included into presentations.
+ *Responsive Design*: The slide viewer adapts to different screen sizes and input modalities.
+ *Interactive Navigation*: Slides can be navigated using keyboard controls, as well as touch, tilt, shake, and swipe gestures.
+ *Live Code Integration*: Offers embedding of live code.
+ *Figure Zooming and Panning*: Includes zoom and pan features for enhanced image interaction.
+ *Export as PDF*: Print-based export with extended print settings.

Limiations of Rslidy include:

- *Collaboration*: No built-in support for collaboration.
- *Export as PowerPoint*: Not possible to export the presentation as PowerPoint.

```
<section>
  <h1>Rslidy Example Slide</h1>
  <div class="columns">
    <ul class="item-auto">
      <li>Bullet Point 1</li>
      <li>Bullet Point 2</li>
      <li>Bullet Point 3</li>
    </ul>
    <figure>
      <div class="figure" style="width:100%;">
        <img src="diagrams/diagram.svg" alt="Example Image" />
      </div>
    </figure>
  </div>
</section>

<style>
  .columns {
    display: flex;
    justify-content: space-between;
    align-items: flex-start;
    gap: 1.25rem;
  }
  .item-auto {
    flex: 1;
    margin-top: 3rem
  }
  figure {
    flex: 1;
    margin: 0;
  }
</style>
```

Listing 3.12: Rslidy: Example slide source code. The resulting slide is shown in Figure 3.23.

Chapter 4

Rslidy v1

Rslidy v1 is a responsive HTML-based slide deck tool, which was inspired by Slidy2 [Raggett 2006]. The project originated in 2013, as part of the course Information Architecture and Web Usability at Graz University of Technology.

Patrick Hipp took on the project and built the first major release of Rslidy, Rslidy v1, for his Master's thesis [Hipp 2021a; Hipp 2021b]. Rslidy v1 provides a modular architecture for slide rendering, navigation, and user interaction, while adding responsive layout behaviour across different devices and display sizes. The tool demonstrates that presentations can be designed, authored, and rendered entirely using open web technologies. Implemented with HTML, CSS, and TypeScript, Rslidy v1 offers a lightweight, browser-native environment suitable for any modern web browser. The user interface of Rslidy v1 can be seen in Figure 4.1, with an example slide showing a data table; the Print Settings panel is also visible.

4.1 General Architecture

The architecture of Rslidy v1 follows the modular design described by Hipp in his thesis [Hipp 2021b, Chapter 5]. The system is organised as a collection of loosely coupled components, each responsible for a specific aspect of slide rendering, navigation, or user interaction. This modular structure supports responsive behaviour, accessibility features, and input methods, while enabling individual components to be developed and maintained independently.

At an architectural level, Rslidy v1 separates slide content, visual presentation, and interactive behaviour using HTML, CSS, and JavaScript. This separation is reflected in the project's directory structure and internal module dependencies. The architecture can be characterised by the following three functional responsibilities:

- *Document Structure*: Slides are authored in plain HTML, with each slide represented by a `<div class="slide">` or `<section>` element. The HTML document contains the complete presentation content and metadata.
- *Presentation and Layout*: Visual appearance and layout behaviour are handled by a set of CSS files, including `rslidy.css`, `toolbar.css`, and `overview.css`. These stylesheets define typography, positioning, transitions, and responsive behaviour, and can be customised independently.
- *Control and Interaction*: Interactive behaviour is implemented through JavaScript modules which manage navigation, keyboard and touch input, incremental content display, and auxiliary features such as the slide overview. Interaction is realised by observing user events and updating the Document Object Model (DOM) through the application of CSS classes.

During initialisation, Rslidy v1 scans the document for slide elements and builds an internal structure to track the currently active slide. Additional modules dynamically generate overview views and manage

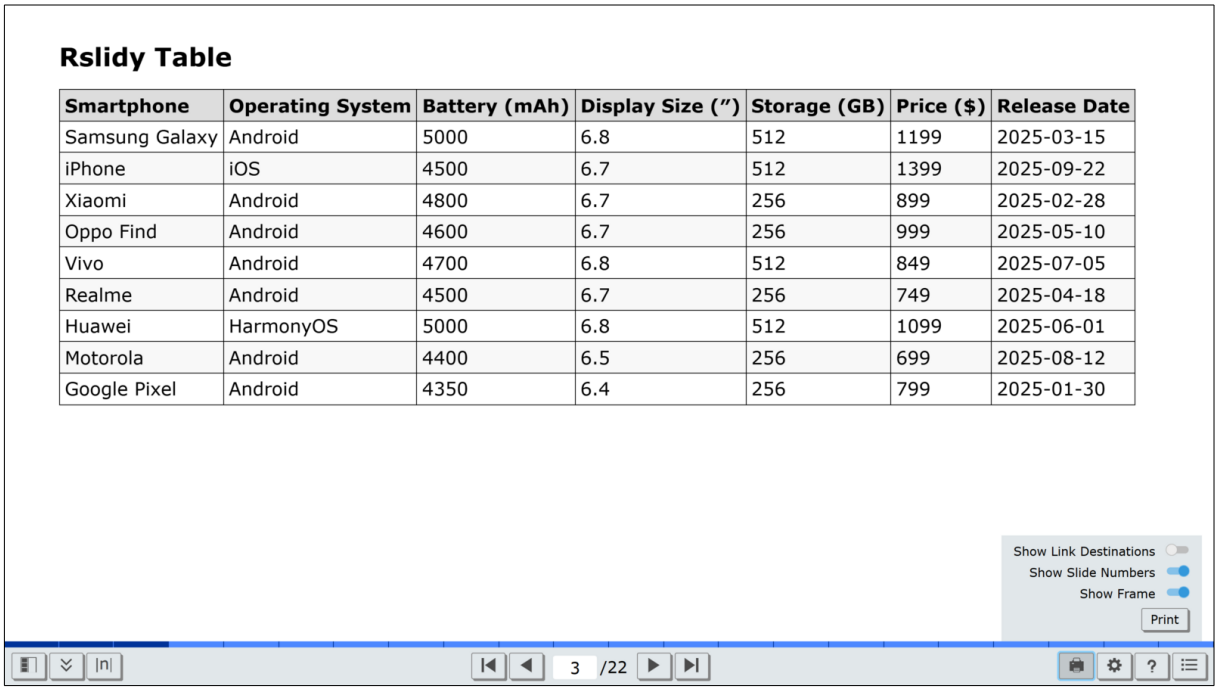


Figure 4.1: Rslidy v1: The default visual appearance and user interface, including the Print Settings panel. The example slide contains a data table.

incremental content reveals. This structure enables complete presentations to be contained in a single HTML file, while maintaining modularity through external style and script resources. An overview of the module structure and internal dependencies of Rslidy v1 is shown in Figure 4.2.

4.2 Styling and Configuration Variables

Rslidy v1 introduces a CSS variable system that centralises key parameters such as colours, fonts, and layout metrics within a single `variables.css` file. This structure forms the visual foundation of the tool, ensuring consistent styling across all components and enabling easy customisation by redefining variables without modifying the base CSS.

The styling architecture follows a modular, inheritance-based approach. Core layout and behaviour definitions are distributed across multiple CSS source files, which are combined into a single `rslidy.css` stylesheet during the build process. This combined file serves as the primary styling entry point at runtime. Supplementary stylesheets extend these rules to refine specific aspects.

By separating presentation logic from visual configuration, Rslidy v1 simplifies maintenance and supports long-term adaptability. Theme adjustments can be achieved entirely through variable overrides or style extensions, reducing redundancy and ensuring a clear hierarchy between core design rules and user-defined styling.

4.3 Navigation and Interaction

Rslidy v1 provides a navigation and interaction model that supports keyboard input, on-screen toolbar controls, mouse interaction, and touch gestures. Slide navigation is realised through animated slide transitions, supporting both sequential movement through the presentation and direct access to individual slides. All interaction methods are processed by a central JavaScript controller, which maps user input to changes in slide state across different devices.

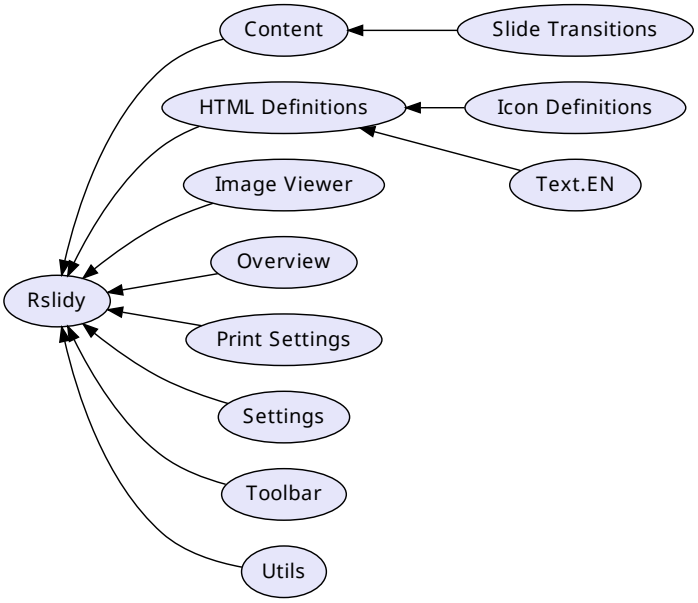


Figure 4.2: Rslidy v1: Module structure and internal dependencies. [Extracted from Hipp [2021b, Figure 5.1] and used under the terms of a CC BY 4.0 licence.]

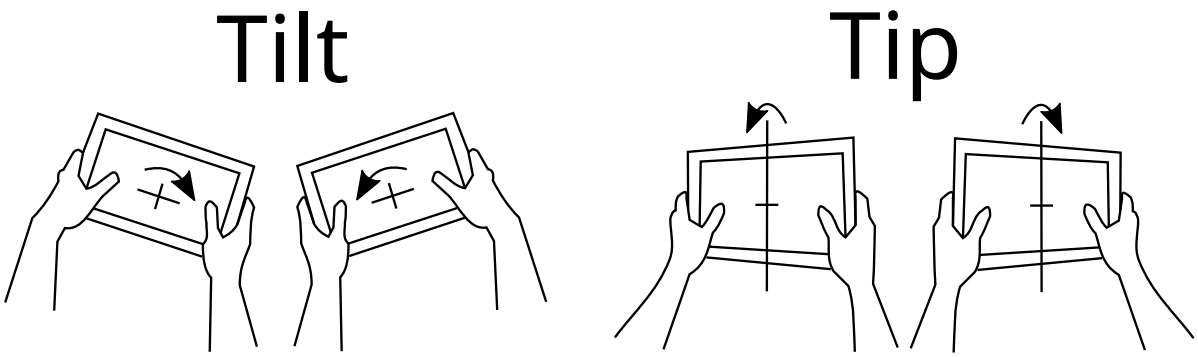


Figure 4.3: Tilt and tip gestures can be performed by either tilting or tipping the device towards the left or right side. [Adapted from Hipp [2021b, Figure A.1] and used under the terms of a CC BY 4.0 licence.]

On-screen controls are aligned with keyboard-based interaction, ensuring that core navigation and presentation functions remain accessible regardless of the chosen input method. In addition to slide navigation, the interaction model includes mechanisms for adjusting font sizes and accessing auxiliary controls during a presentation. This design enables presenters to adapt both content and interface elements to different viewing contexts without interrupting the presentation flow.

Alongside keyboard and mouse-based navigation, tilt and tip gestures provide an intuitive interaction mechanism on touch-capable devices, complementing traditional input methods and supporting use on tablets and similar devices. An illustration of the tilt and tip gestures can be seen in Figure 4.3.

4.4 Responsive Design Implementation

Rslidy v1 applies a responsive layout strategy based on CSS Flexbox and media queries to maintain visual consistency across different devices and display sizes. The design automatically scales slide content to fit the available viewport, while preserving proportions and ensuring that text and visual elements remain readable.

Rslidy primarily uses relative units such as percentages and `em` or `rem` values to support proportional scaling, and uses CSS transitions to ensure smooth layout adjustments when resizing the browser window or changing device orientation. While these mechanisms provide a robust foundation for responsive behaviour, the actual responsiveness of a presentation depends on how content is authored. Slide creators are responsible for applying appropriate Rslidy classes, avoiding custom CSS rules that override responsive behaviour, and ensuring that their slide content remains responsive across different screen sizes.

4.5 Limitations and Motivation for Further Development

While Rslidy v1 provides a stable foundation for HTML-based presentations, several technical and functional limitations motivated further development. Dependency management through `npm` [NPM 2026a] can result in redundant packages and non-deterministic builds, making it difficult to maintain consistent development environments and workflows.

Another limitation lies in the printing process, which relies largely on browser-specific rendering behaviour. This can lead to irregular margins, scaling issues, or truncated slides when exporting to PDF or printing, making it challenging to achieve predictable results across different browsers and systems.

Responsiveness is also limited for complex structures such as wide tables or dense data layouts, which often require manual adjustment. These constraints reduce the overall adaptability of the tool across device types, screens, and PDF output. In addition, the lowlight mode requires manual activation and does not integrate with system-level theme preferences or browser settings, highlighting the need for more flexible light-dark mode adaptation mechanism.

Collectively, these limitations underline the need for further development to improve maintainability, print output consistency, and responsive behaviour, motivating an updated iteration of Rslidy.

Chapter 5

Rslidy v2

Rslidy v2 extends the original framework developed by Patrick Hipp with a modernised architecture and improved support for responsive presentations. The latest version developed as part of this thesis is released as Rslidy version 2.1.0 [Platzer 2026]. While preserving the lightweight, browser-native design of earlier versions, this release introduces significant improvements to the development workflow, printing, theming, and responsive content handling.

Key changes include a deterministic build setup, extended print configuration, automatic light-dark theme adaptation, and consistent responsive behaviour across devices. The transition from the traditional NPM package manager [NPM 2026a] to PNPM [PNPM 2026], together with the adoption of TypeScript, aligns Rslidy v2 with current front-end development practices. Figure 5.1 illustrates an example slide created in Rslidy v2, featuring an updated responsive data table and the revised Print Settings panel.

5.1 System Modernisation

A central part of the modernisation of Rslidy v2 is the revision of the build workflow. Earlier versions of the framework relied on `NPM` for dependency management. Although widely used, its default dependency resolution strategy is based on a shared `node_modules/` folder, which can result in duplicated packages and less predictable installations.

During the transition to a more reliable build setup, two package managers were evaluated: `Yarn` and `PNPM`. Yarn [Yarn 2026] was considered due to its improved performance characteristics and stricter lockfile handling compared to NPM. It also introduces advanced concepts such as Plug'n'Play and zero-install workflows. However, these features rely on non-standard dependency resolution mechanisms and increase complexity. In addition, `Yarn` continues to duplicate dependencies when using a traditional `node_modules/` layout, which does not sufficiently address long-term efficiency and storage concerns.

Based on this evaluation, Rslidy v2 now uses `PNPM` as its package manager [PNPM 2026]. PNPM introduces content-addressable storage and isolated `node_modules/` directories by default, ensuring that dependencies are stored only once and linked into individual projects. This avoids redundant installations, reduces disk usage, and ensures that all contributors resolve an identical dependency tree. Additional features such as automatic peer dependency installation, dependency patching, and side-effects caching further support deterministic and maintainable builds.

Task automation is still managed with Gulp, which handles file watching, bundling, and minification. In addition, the TypeScript setup was updated, providing improved static type checking and contributing to better code clarity and long-term maintainability. Together, these changes establish a more predictable, efficient, and robust development workflow. Figure 5.2 summarises the build-time structure of Rslidy v2, showing how package metadata, TypeScript sources, stylesheets, and Gulp-based processing are combined

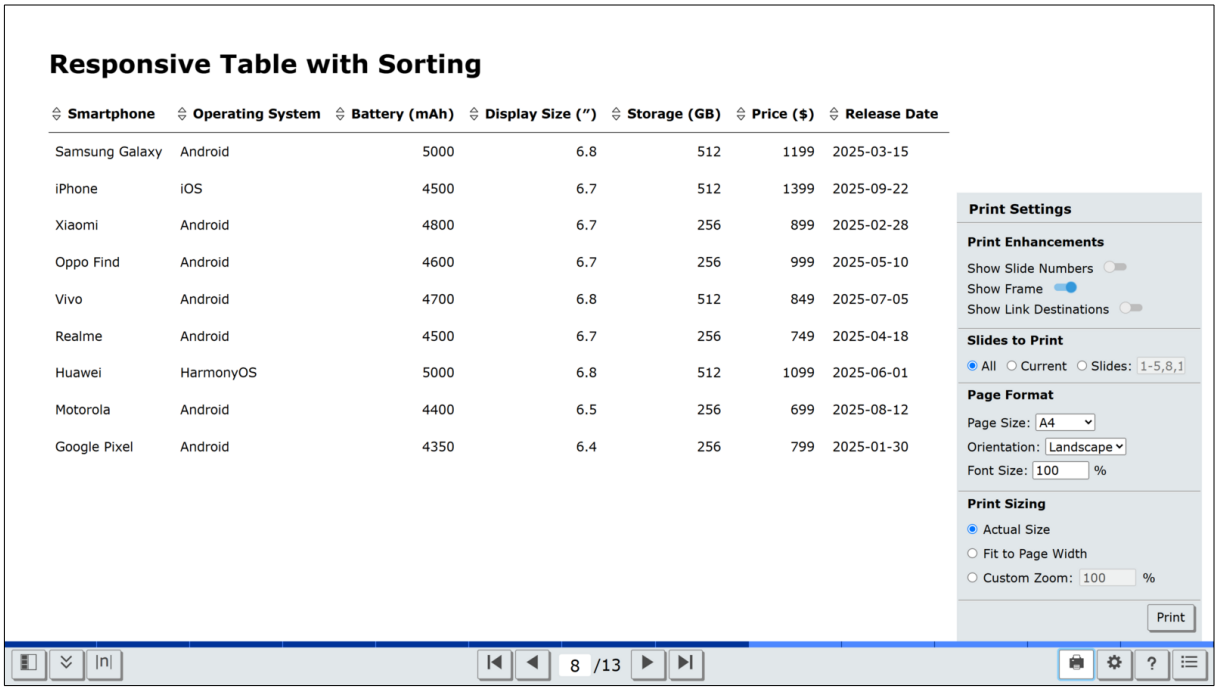


Figure 5.1: Rslidy v2: User interface and expanded Print Settings panel. The example slide contains a data table, which Rslidy v2 makes sortable by default.

into the emitted JavaScript and CSS artefacts. Selected parts of the resulting package configuration are shown in Listing 5.1.

Beyond the build workflow, Rslidy v2 also reorganises the runtime viewer into a clearer set of TypeScript modules. The entry module initialises the presentation, injects the user interface, and coordinates specialised components for navigation, settings, printing, overview rendering, content handling, and image viewing. Figure 5.3 provides an overview of these runtime components and their responsibilities within the viewer architecture.

5.2 Updated Print Settings

Printing support was extended in Rslidy v2 to provide more reliable and predictable PDF output across browsers and devices. While Rslidy v1 already offered a Print Settings panel with basic options such as controlling the display of link destinations, slide numbers, and slide frames, other aspects including page size, orientation, and scaling were largely left to the browser's native print behaviour. This could result in inconsistent margins or clipped content.

Rslidy v2 preserves the existing panel and the runtime generation of print-specific CSS, but extends both with additional functionality. Users can now configure parameters such as paper format, page orientation, font scaling, and slide selection, allowing the output to be adapted to different requirements. To control how slide content is placed on the printed page, three sizing modes are provided: Actual Size, Fit to Page Width, and Custom Zoom. These options allow the output to be adapted to different presentation and documentation requirements without relying on browser-specific print settings.

All configuration changes are applied immediately by regenerating the print stylesheet, enabling users to preview the effect before exporting to PDF or printing. This approach maintains the lightweight, browser-native workflow of the original implementation, while offering greater flexibility and more consistent results.

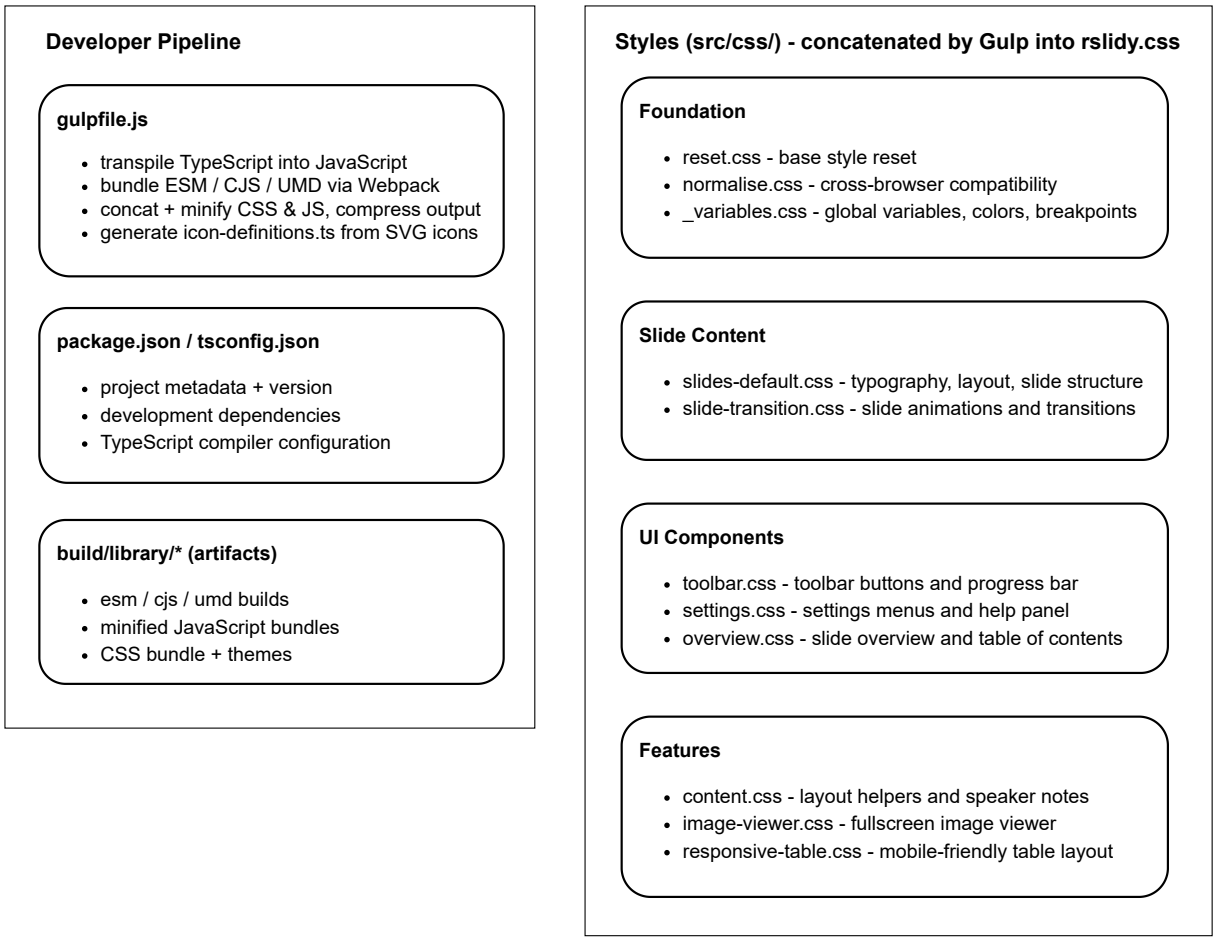


Figure 5.2: Build-time overview of Rslidy v2. TypeScript sources and CSS stylesheets are processed into artefacts.

```
{
  "name": "rslidy",
  "version": "2.1.0",
  "description": "Rslidy: Lightweight, Accessible, and Responsive HTML Slide Decks",
  "main": "build/rslidy.js",
  "devDependencies": {
    "gulp": "^4.0.2",
    "gulp-typescript": "^6.0.0-alpha.1",
    "typescript": "^5.7.3",
    "webpack": "^5.89.0",
    "webpack-stream": "^7.0.0"
  }
}
```

Listing 5.1: Rslidy v2: Simplified excerpt from the package configuration, showing project metadata and selected development dependencies used by the build workflow.

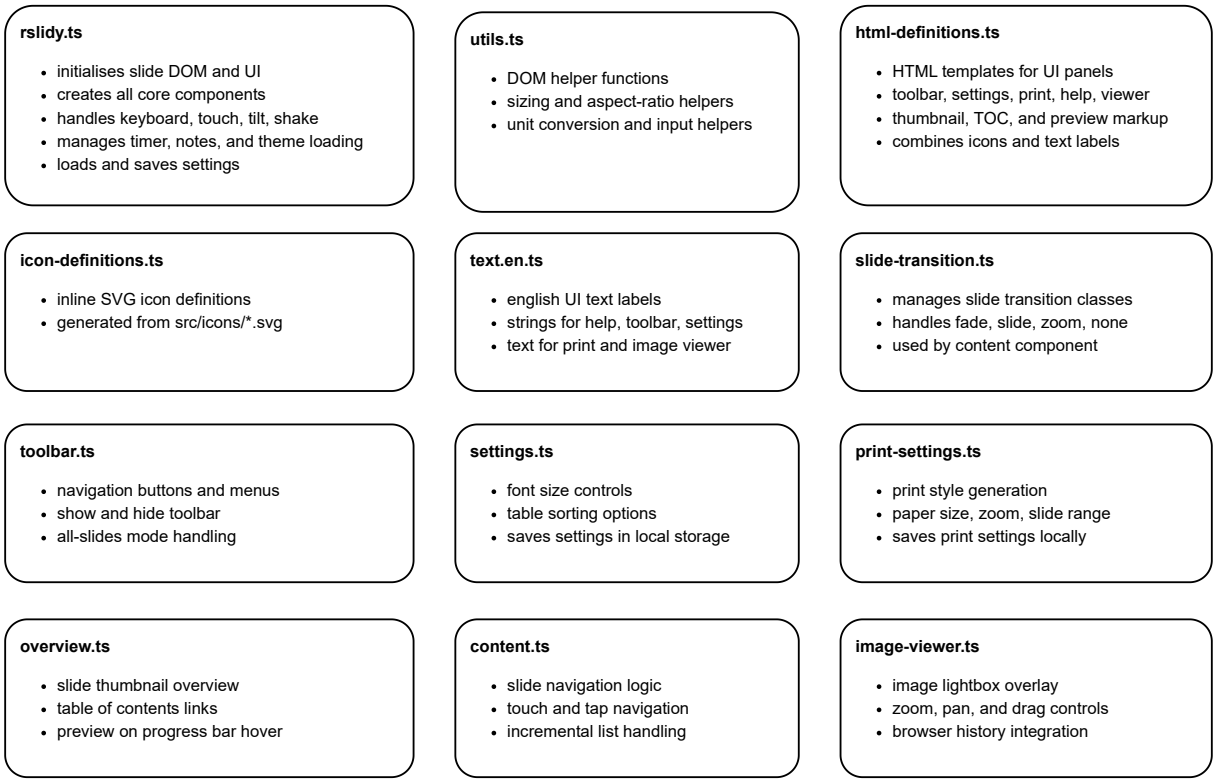


Figure 5.3: Runtime components of Rslidy v2, showing the main TypeScript viewer modules and their responsibilities within the Rslidy v2 runtime.

5.3 Light–Dark Mode Integration

To reflect current accessibility and usability standards, Rslidy v2 now implements a dynamic colour system using the CSS `light-dark()` function. Rather than requiring manual switching, the presentation theme automatically adapts to the user's operating system or browser preference, applying light or dark colours as appropriate.

All key colours are defined as custom CSS properties prefixed with `--rslidy-*` in the global `:root` scope. These variables are resolved via `light-dark()` expressions, allowing both light and dark variants to be derived from a single declaration, ensuring consistent contrast and readability across themes. This also simplifies future theming or branding, as colour palettes can be replaced without altering the structural CSS. In addition to general theme switching, several slide-specific elements such as toolbars, frames, and SVG-based icons automatically adjust to the current theme. These improvements eliminate the need for duplicate style declarations and enhance maintainability.

The use of the CSS `light-dark()` function further allows dynamic updates at runtime. When the browser or operating system changes its colour-scheme preference, Rslidy reacts immediately without requiring a page reload, maintaining a seamless experience during live presentations. This reactive behaviour demonstrates how declarative CSS logic and modern browser features can be combined to achieve accessibility without JavaScript intervention. An example of a dark-mode presentation can be seen in Figure 5.4.

5.4 Responsive Tables

Rslidy v2 extends the behaviour of conventional HTML tables and images to better support responsive presentation contexts. Standard `<table>` elements are mainly designed for wide layouts and offer limited

Figure 5.4: Rslidy v2: Dark-mode presentation view, illustrating automatic theme adaptation based on the user's system preferences. Colour values are derived using the CSS `light-dark()` function.

adaptability on small screens. Tables defined with the `rslidy-responsive-table` class now support horizontal scrolling, dynamic text scaling, and semantic alignment of column types. Supplementary table classes extend this behaviour by allowing finer control over typography and layout alignment where required. Interactive sorting can be triggered directly via header clicks, with visual feedback shown through ascending or descending icons. These features enable tables to remain readable on smaller screens while offering interactivity during presentations. Further details of the responsive table implementation are provided in Chapter 6.

5.5 Summary

Rslidy v2 provides a modern, flexible foundation that merges modern web standards with a minimalist design philosophy [MDN 2025f]. The move to PNPM ensures deterministic, maintainable builds, while the improved print and theming implementations deliver reliable results in every environment. Responsive tables, automatic theme adaptation, and reliable media scaling make the system more robust and accessible. These architectural refinements provide a stable foundation for future extensions, such as plugin-based customisation, dynamic content generation, and collaborative editing features for live presentations.

Chapter 6

Selected Details of the Implementation

Certain components of Rslidy v2 were particularly tricky to implement and required multiple iterations and experimentation to get right. Two of these components are described in more detail here: enhanced print settings and support for responsive tables. Both required extensive refinement to achieve consistent performance across browsers, screen sizes, and print outputs.

6.1 Enhanced Print Settings

In Rslidy v1, print support was limited to three visual toggles: Show Link Destinations, Show Slide Numbers, and Show Frame. While these options addressed basic presentation requirements, they did not provide control over page layout, slide selection, or scaling behaviour, and therefore relied heavily on the browser's default print handling.

Rslidy v2 introduces a redesigned Print Settings panel that extends print and PDF export functionality with additional layout and scaling controls. Slide layout, sizing, and visibility can be adjusted directly within the presentation interface. The final rendering is prepared explicitly by applying dynamically generated, print-specific CSS rules at runtime. The updated print settings provide control over page format and orientation, global font scaling, slide selection, and print sizing behaviour. Slides can be printed as a complete deck, as the currently active slide, or as a user-defined range. In addition, multiple sizing modes allow slide content to be preserved at its original scale, adapted to the printable page width, or manually scaled using a zoom factor.

Together, these extensions improve the predictability and consistency of printed and exported PDF results across different browsers and devices, while keeping all print-related configuration local to the slide deck interface.

6.1.1 Redesigned Print Settings Panel

The controls in the Print Settings panel allow users to adjust page layout, orientation, font size, slide selection, and element visibility directly within the presentation. All selected values are processed at runtime and translated into dynamically generated CSS rules that are applied before printing.

The redesigned Print Settings panel groups related options for page setup, slide selection, and print sizing, as can be seen in Figure 6.1. In addition to the original visual toggles, the panel provides controls for paper format, orientation, font scaling, slide selection, and sizing behaviour. The user can select between different paper formats and orientations through the Page Size and Orientation selectors. Supported paper sizes include standard values such as A4, Letter, or A5, while orientation can be toggled between Portrait and Landscape. Additionally, the Font Size input provides a global text scaling factor in percent for the printed output.

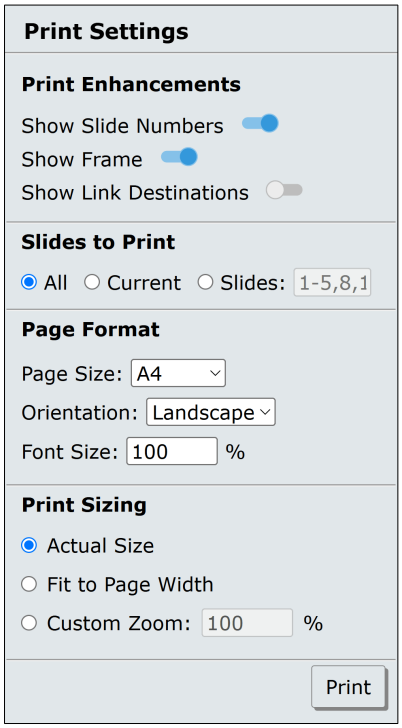


Figure 6.1: New Print Settings panel with controls for layout, orientation, paper size, font scaling, and slide visibility. Users can choose between three sizing options, Actual Size, Fit to Page Width, or Custom Zoom, and toggle elements like link destinations, slide numbers, and frames.

With the option Slides to Print, users can print all slides, only the current slide, or a custom range such as "1–5". This range input is parsed and converted into display rules that selectively show or hide slides. The implementation of this functionality is shown in Listing 6.1. Print sizing is controlled by a group of radio buttons representing different modes: Actual Size, Fit to Page Width, and Custom Zoom. The Actual Size option preserves the slide layout as rendered on screen, allowing the browser's print engine to handle pagination based on the selected paper format. The Fit to Page Width option constrains slides to the available page width while maintaining their internal layout and aspect ratio. This reduces the risk of clipped content or horizontal overflow. The Custom Zoom option applies a user-defined scaling factor to slide content, allowing fine-grained adjustment of the balance between content density and readability.

The three original optional checkboxes provide additional visual control during printing. By enabling Show Slide Numbers, an automatically incremented counter is appended to each slide. Activating Show Link Destinations renders visible URLs after hyperlinks, while Show Frame draws a border around each slide to visually distinguish individual slides in multi-page outputs.

6.1.2 Dynamic CSS Generation

Rslidy v2 generates all print-related CSS dynamically at runtime. Rather than maintaining a separate print stylesheet, the active configuration is translated into a single @media print rule set whenever the user changes a print setting.

Each interaction with the Print Settings panel triggers regeneration of a dedicated style element. Before a new configuration is applied, any previously injected print stylesheet is removed from the document head. This ensures that outdated rules do not persist across multiple print attempts and that the generated CSS always reflects the current print configuration. The runtime generation process combines paper format, orientation, font scaling, slide visibility rules, and sizing mode into a single CSS definition. Page-level properties are expressed via @page, while slide-level adjustments are applied through targeted selectors.

```
private applyCustomSlideRange(range: number[]): string {
  const slides =
    document.querySelectorAll("#rslidy-content-section .slide");

  const valid = range.filter(
    n => n >= 1 && n <= slides.length
  );

  const visible = valid
    .map(num =>
      `#rslidy-content-section .slide:nth-of-type(${num}) {
         display: block !important;
       }`
    )
    .join("\n");

  return `
    #rslidy-content-section .slide {
      display: none !important;
    }
    ${visible}
  `;
}
```

Listing 6.1: Generating CSS selectors to hide all slides by default and selectively re-enable only the slides specified by a user-defined range for printing.

The resulting stylesheet is injected into the document head immediately before printing. Listing 6.2 shows a shortened excerpt of this mechanism, illustrating how page setup and font scaling are mapped to print-specific CSS without modifying the underlying slide markup.

Since all print behaviour is expressed through generated CSS, no DOM restructuring or content duplication is required. Slide selection, scaling, and visibility rules are applied exclusively at print time, keeping the on-screen presentation unaffected. To improve usability across sessions, the current print configuration is persisted using the localStorage API. When the Print Settings panel is reopened, previously stored values are restored and immediately applied, allowing users to reproduce consistent print results without manual reconfiguration.

6.2 Responsive Tables

The responsive table implementation was redesigned in Rslidy v2 to provide improved readability, better accessibility, and enhanced interactivity across both screen and print contexts. Tables in Rslidy v2 can be made responsive and sortable by applying the class `rslidy-responsive-table` to the `<table>` element. Earlier versions relied on static table markup without sorting or adaptive layouts, which limited the clarity of wider tables on narrower devices. The revised implementation introduces automatic labelling for narrow containers, dynamic column sorting, and context-sensitive style adaptations based on container size queries. This approach allows tables to adapt to the width of their surrounding layout region instead of relying on global viewport breakpoints.

```
private applyPrintSettings() {
  if (this.style)
    document.head.removeChild(this.style);

  const layout = this.view.querySelector(
    "#rslidy-select-orientation"
  ) as HTMLSelectElement;

  const paper = this.view.querySelector(
    "#rslidy-select-paper-size"
  ) as HTMLSelectElement;

  const font = this.view.querySelector(
    "#rslidy-input-font-size"
  ) as HTMLInputElement;

  let css = "@media print {\n";
  css += `@page { size: ${paper.value} ${layout.value}; }\n`;
  css += `body { font-size: ${font.value}%; }\n`;
  css += "}\n";

  this.style = document.createElement("style");
  this.style.innerHTML = css;
  document.head.appendChild(this.style);
}
```

Listing 6.2: Creating and injecting a print-specific stylesheet based on the current paper format, orientation, and font scaling settings.

6.2.1 Layout and Structure

The base styling is defined by the `rslidy-responsive-table` class. Tables using this class are designed to respond dynamically to the available width of their parent container while preserving horizontal scrolling when required. If the container becomes too narrow to display the full tabular layout, the table automatically switches to a stacked representation while keeping all content accessible. Spacing and typography are reduced to 70% of the slide font size to improve information density, while visual clutter is minimised by suppressing explicit borders.

The complementary `rslidy-responsive-table-text-scaling` class applies dynamic font scaling through the `clamp()` function, ensuring that text remains readable across varying container sizes and print layouts. This adaptive scaling mechanism also improves readability on smaller screens by proportionally adjusting the cell content without altering table geometry. Optional zebra-striping can be enabled by adding the `rslidy-zebra` class, which applies alternating row colours for improved visual distinction. Column alignment is handled through semantic class names: `rslidy-numeric` aligns numbers to the right, `rslidy-symbol` centres icons and symbols, and `rslidy-text` left-aligns textual cells.

To simplify table markup, these classes only need to be applied to the corresponding `<th>` elements. Rslidy v2 automatically transfers the alignment classes to the associated `<td>` elements at runtime, ensuring consistent alignment across the complete column. Together, these classes ensure that tabular content remains visually coherent and accessible across all screen sizes, devices, and PDF and print outputs.

The code for an example slide containing a responsive table is shown in Listing 6.3. On a wider screen,

```
<section>
  <h1>Responsive Table with Sorting</h1>
  <table class="rslidy-responsive-table rslidy-responsive-table-text-scaling">
    <thead>
    <tr>
      <th scope="col" class="rslidy-text">Smartphone</th>
      <th scope="col" class="rslidy-text">Operating System</th>
      <th scope="col" class="rslidy-numeric">Battery (mAh)</th>
      <th scope="col" class="rslidy-numeric">Display Size ('')</th>
      <th scope="col" class="rslidy-numeric">Storage (GB)</th>
      <th scope="col" class="rslidy-numeric">Price ($)</th>
      <th scope="col" class="rslidy-text">Release Date</th>
    </tr>
    </thead>
    <tbody>
    <tr>
      <td>Samsung Galaxy</td>
      <td>Android</td>
      <td>5000</td>
      <td>6.8</td>
      <td>512</td>
      <td>1199</td>
      <td>2025-03-15</td>
    </tr>
      ...
    <tr>
      <td>Google Pixel</td>
      <td>Android</td>
      <td>4350</td>
      <td>6.4</td>
      <td>256</td>
      <td>799</td>
      <td>2025-01-30</td>
    </tr>
    </tbody>
  </table>
</section>
```

Listing 6.3: Rslidy slide containing a responsive table with sortable columns demonstrating the `rslidy-responsive-table` layout. The corresponding wide table layout is shown in Figure 6.2, while the transformed narrow layout is shown in Figure 6.3.

the columns are presented next to one another in a classic tabular view, as shown in Figure 6.2. On a narrow screen, tables automatically transition into a stacked layout using CSS Flexbox, as can be seen in Figure 6.3. In this mode, the table header is hidden, and each cell is preceded by a label derived from the original column heading. This label is inserted using the CSS `content: attr(data-label)` property and styled to appear on the left side of each cell, forming a readable key value structure.

6.2.2 Automatic Data Labels

To support stacked table layouts on narrow viewports and in constrained print contexts, Rslidy automatically generates header-based data labels at runtime. These labels provide the textual context required to interpret individual cell values once the table header is hidden. Label generation is implemented by the `applyResponsiveTableLabels()` function. It scans all tables marked with the `rslidy-responsive`

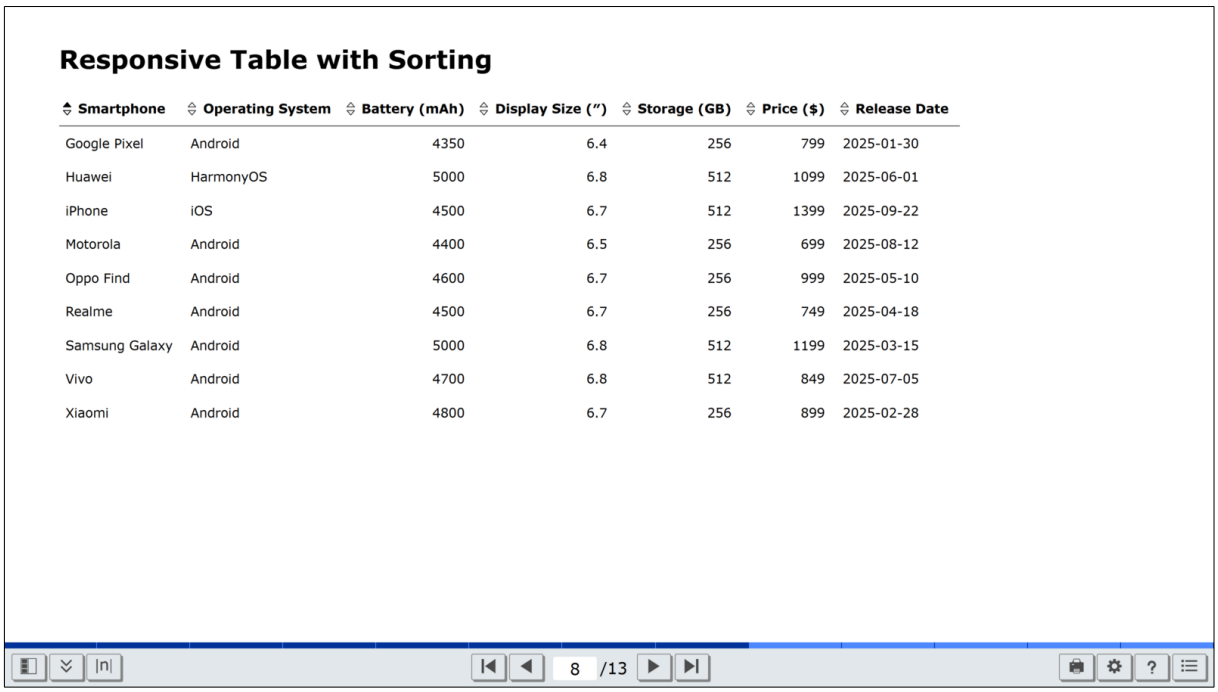


Figure 6.2: Responsive table example slide on a wider screen. The class `rslidy-responsive-table` is used to make the table responsive and enable column-based sorting. The class `rslidy-responsive-table-text-scaling` is used to automatically scale table text to fit the available space. The corresponding HTML source code is shown in Listing 6.3.

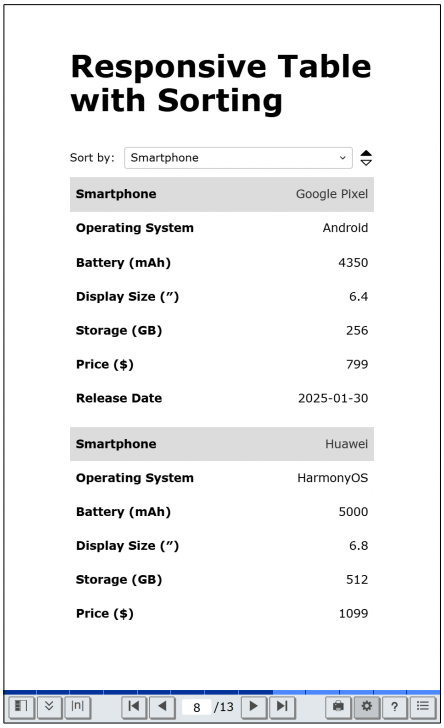


Figure 6.3: Responsive table example on a narrower screen, demonstrating the stacked layout transformation of the responsive table. At reduced widths, table headers are converted into `data-label` attributes displayed before each cell. The corresponding HTML source is shown in Listing 6.3.

```
applyResponsiveTableLabels(): void {
  const tables = document.querySelectorAll("table.rslidy-responsive-table");

  tables.forEach(table => {
    const headers = Array.from(table.querySelectorAll("thead th"))
      .map(th => th.textContent?.trim() || "");

    const rows = table.querySelectorAll("tbody tr");

    rows.forEach(row => {
      const cells = row.querySelectorAll("td");

      cells.forEach((cell, index) => {
        if (!cell.hasAttribute("data-label") && headers[index]) {
          cell.setAttribute("data-label", headers[index]);
        }
      });
    });
  });
}
```

Listing 6.4: Automatically assigning header text to data-label attributes for responsive table rendering.

`-table` class, extracts the text content of each header cell, and assigns the corresponding values to `data-label` attributes on the table body cells. This mapping preserves the column–cell relationship independently of the visual layout used for rendering.

The function is executed automatically after the slide content has been loaded, ensuring that both static and dynamically inserted tables are annotated consistently. Existing `data-label` attributes are left unchanged, allowing manual overrides where required. The implementation for adding the data labels is shown in Listing 6.4. By deriving labels directly from the table header, this approach keeps the rendered table content synchronised with its underlying column structure. A single table definition can therefore transition reliably between wide, scrollable layouts and stacked, key-value layouts without requiring duplicate elements or layout-specific adaptations.

6.2.3 Interactive Sorting

Responsive tables in Rslidy support interactive column sorting to improve the handling of larger datasets. Sorting is initialised by the `setupTableSorting()` function, which registers event listeners on header cells once the slide content has been loaded. Each sortable column header is augmented with an SVG icon that indicates the current sorting state. The sorting icon is positioned inside the header cell using CSS pseudo-elements and visually communicates the current ordering mode to the user.

Figure 6.4 illustrates the appearance of sorting icons in a header row. In its default state, the icon shows a neutral indicator suggesting that sorting is available. After user interaction, the icon changes to reflect either ascending or descending ordering. This visual feedback is implemented by toggling the CSS classes `sorted-asc` and `sorted-desc`, which replace the base icon with a variant indicating the active sort direction.

User interaction follows a defined sequence. Selecting a header applies ascending order, a second selection switches to descending order, and a third selection restores the original row order. The active

Figure 6.4: Table header row used in sortable responsive tables.

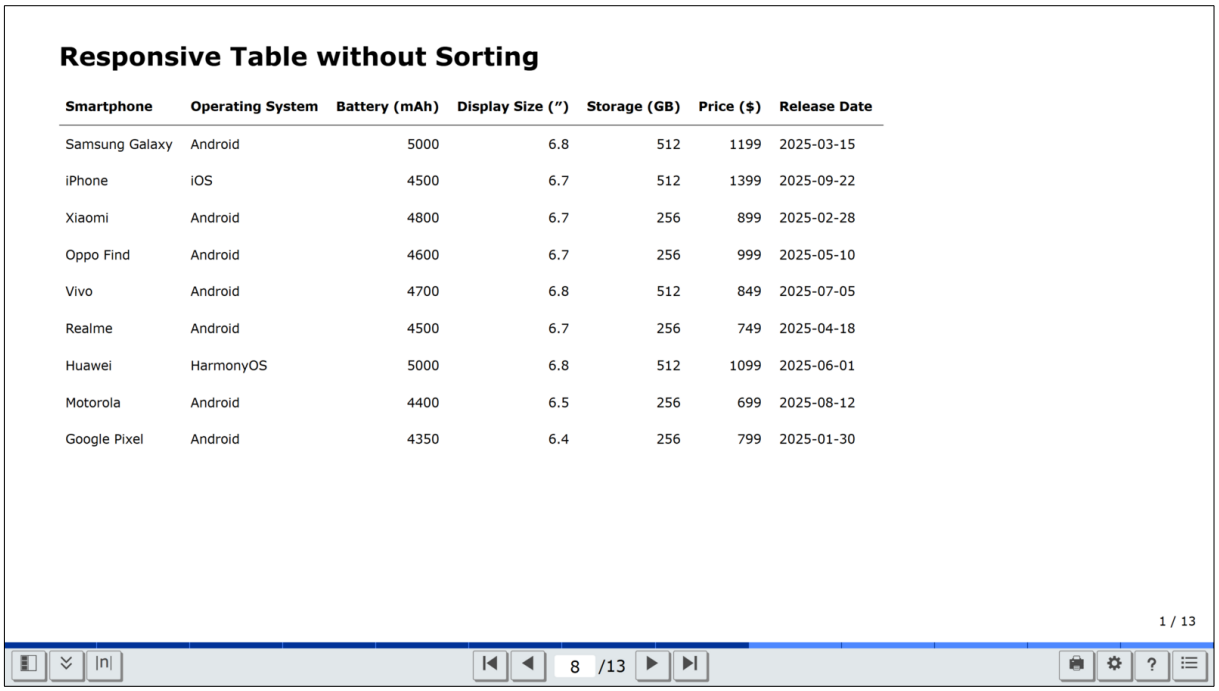


Figure 6.5: Responsive table without sorting.

column index and sort direction are stored per table instance, allowing the sorting state to remain consistent when switching between desktop and mobile layouts.

Before comparison, cell values are normalised based on their content. Numeric values, symbols such as checkmarks, and date strings are handled explicitly to ensure correct ordering. Sorting is applied by rearranging the existing rows within the `<tbody>` element, avoiding any structural changes to the table or its surrounding container.

Sorting is automatically disabled for empty headers and for columns marked with the `rslidy-no-sort` class. A responsive table without interactive sorting is shown in Figure 6.5. Visual feedback is implemented through the CSS classes `sorted-asc` and `sorted-desc`, which update the SVG icon embedded in the corresponding header to reflect the active sorting direction. The code responsible for toggling the sort direction and reordering the table rows is shown in Listing 6.5. It illustrates the core comparison and row-reordering logic applied during a sort operation. In combination with responsive layout handling and automatic data labels, interactive sorting ensures that tabular data remains usable and consistent across different screen sizes and print contexts without requiring additional configuration.

```
const sortTable = (
  tbody: Element,
  columnIndex: number,
  direction: "asc" | "desc"
) => {
  const rows = Array.from(tbody.querySelectorAll("tr"));

  rows.sort((a, b) => {
    const cellA = a.children[columnIndex]?.textContent || "";
    const cellB = b.children[columnIndex]?.textContent || "";

    const valA = normalize(cellA);
    const valB = normalize(cellB);

    if (typeof valA === "number" && typeof valB === "number") {
      return direction === "asc" ? valA - valB : valB - valA;
    }

    return direction === "asc"
      ? String(valA).localeCompare(String(valB))
      : String(valB).localeCompare(String(valA));
  });

  tbody.innerHTML = "";
  rows.forEach(row => tbody.appendChild(row));
};
```

Listing 6.5: The code to apply the selected sort direction to a table column.

Chapter 7

Outlook and Future Work

Rslidy v2 introduces a deterministic build pipeline, modular TypeScript architecture, extended print configuration, responsive media handling, and automatic theme adaptation. The framework now provides a stable and modern foundation for browser-native slide decks. Despite these improvements, several architectural and functional extensions remain open. The following sections outline concrete development directions to extend the current implementation, without compromising its lightweight and standards-driven design.

7.1 Deterministic PDF Export

Rslidy v2 dynamically generates print-specific CSS based on user-selected configuration parameters. Page format, scaling behaviour, and slide selection are translated into `@media print` rules at runtime. This improves flexibility, but final PDF output still depends on the rendering behaviour of the user's browser.

A deterministic export pipeline could extend the existing print subsystem by executing slide rendering inside a controlled headless environment during build time. A scripted export command could load the slide deck with predefined viewport dimensions and standardised print parameters before generating a reproducible PDF artefact. Such a workflow would ensure consistent pagination, scaling behaviour, and margin calculation, independent of client-side browser configuration. A conceptual workflow for deterministic PDF export using a headless browser is shown in Listing 7.1.

7.2 Presenter View

Basic presenter functionality already exists since Rslidy v1. Speaker notes can be accessed in example decks via the N key, and timer-related logic is present in the codebase. However, a dedicated synchronised presenter window with slide preview has not been implemented. A future presenter view could build upon the existing navigation controller and note handling logic. Instead of introducing parallel navigation systems, the main slide state could be serialised and propagated to a secondary window. Since Rslidy v2 already maintains a deterministic slide index and central navigation handler, such synchronisation would extend existing mechanisms rather than duplicate logic. A conceptual state propagation mechanism is shown in Listing 7.2.

```
import { launch } from "headless-browser";

async function exportSlides(): Promise<void> {
  const browser = await launch({
    viewport: { width: 1280, height: 720 }
  });

  const page = await browser.newPage();
  await page.goto("http://localhost:8080/slides.html");

  await page.emulateMediaType("print");

  await page.pdf({
    path: "slides.pdf",
    format: "A4",
    printBackground: true
  });

  await browser.close();
}
```

Listing 7.1: A conceptual workflow for deterministic PDF export using a headless browser, independent of client-side browser behaviour.

```
interface SlideState {
  index: number;
  total: number;
  notes?: string;
}

function broadcastState(state: SlideState): void {
  presenterWindow?.postMessage(state, "*");
}

function onSlideChange(newIndex: number): void {
  const state: SlideState = {
    index: newIndex,
    total: slides.length,
    notes: getSlideNotes(newIndex)
  };

  broadcastState(state);
}
```

Listing 7.2: Propagation of slide state to a secondary presenter window, based on the central navigation controller.

7.3 Slide Virtualisation

In the current implementation, all slides are present in the DOM throughout the presentation lifecycle. Navigation operates by updating active slide classes and maintaining an internal slide index. This model simplifies state handling and ensures predictable behaviour during navigation. However, for very large slide decks containing extensive media content, the permanent presence of all slide nodes may increase memory consumption and initial layout cost.

A slide virtualisation strategy could restrict fully active DOM nodes to the current slide and a limited neighbourhood of adjacent slides. Inactive slides could be detached from the DOM tree or replaced with lightweight placeholders. Upon navigation, required slides would be reattached and reinitialised. Such an approach would require careful state preservation for interactive elements and consistent reattachment logic. The existing modular navigation controller in Rslidy v2 provides a clear insertion point for managing slide lifecycle transitions, making incremental virtualisation technically feasible without rewriting core navigation logic.

7.4 Remote Control Support

Remote navigation would allow presenters to control slides from a secondary device, such as a smartphone. Rslidy v2's centralised slide controller provides a single entry point for navigation events, which simplifies integration of remote commands. A remote control subsystem could transmit navigation actions between devices using either peer-to-peer communication or a temporary relay service. Each navigation command received from a remote device would invoke the same internal methods used for keyboard or toolbar interaction, preserving architectural consistency. Session-based identification mechanisms could restrict access to authorised devices. The deterministic navigation model in Rslidy v2 ensures that externally triggered navigation events remain predictable and synchronised.

7.5 Automated Testing and Continuous Integration

The transition to TypeScript and PNPM in Rslidy v2 improves structural clarity and dependency determinism. Nevertheless, automated testing is currently limited. Future work could introduce unit tests for core subsystems such as navigation handling, responsive table logic, and dynamic print CSS generation. These tests would validate expected behaviour under controlled input scenarios.

In addition, end-to-end tests using a headless browser environment could simulate user interaction sequences, including slide navigation, print configuration changes, and responsive layout adjustments. Integration into a continuous integration pipeline would ensure that each code modification is validated across multiple browser engines before release. This would further strengthen long-term maintainability and support collaborative development.

Chapter 8

Concluding Remarks

Web-based slide decks have become an established alternative to traditional presentation software. A wide variety of tools now exists, ranging from simple Markdown-based systems to feature-rich hosted platforms. Each approach offers different advantages with respect to flexibility, collaboration, usability, and performance. While hosted solutions increasingly dominate the market, due to their collaborative features and integrated analytics, lightweight standalone approaches continue to offer benefits in areas such as runtime efficiency, customisation, and independence from cloud infrastructure.

This thesis examined the landscape of web-based slide deck tools and investigated how modern web technologies can be used to build responsive and adaptable presentation systems. The survey of existing tools highlighted the variety of design philosophies used in current systems, including text-based approaches, JavaScript-based frameworks, and fully hosted presentation platforms. These systems demonstrate different trade-offs between simplicity, extensibility, and runtime capabilities.

Building on this background, the thesis analysed the architecture and design of the Rslidy slide deck framework. Rslidy v1 already demonstrated that responsive slide presentations can be implemented entirely using open web technologies such as HTML, CSS, and JavaScript. At the same time, several technical limitations were identified, particularly regarding print configuration, responsiveness of complex slide elements, and maintainability of the build process.

The main contribution of this thesis is the design and implementation of Rslidy v2. The system introduces several improvements that address the limitations of Rslidy v1, while preserving the lightweight and browser-native philosophy of the original framework. The development environment was modernised using contemporary JavaScript tooling, improving maintainability and ensuring reproducible builds. This restructuring simplifies the development workflow and aligns the project with modern web development practices.

A major improvement introduced in Rslidy v2 is the redesigned print configuration system. Instead of relying entirely on the browser's default print behaviour, print settings can now be configured directly within the presentation interface. Layout parameters such as page format, orientation, slide selection, and scaling are translated into dynamically generated CSS rules. This mechanism allows presentations to produce more predictable and consistent PDF output across different browsers and devices.

Another key contribution is the responsive table implementation. Tables are able to adapt dynamically to the available container size, while preserving readability on smaller screens. Automatic generation of header-based data labels enables tables to transform into a stacked layout when the available space becomes limited. In addition, interactive column sorting allows larger datasets to be explored directly within the presentation. Together, these mechanisms improve the usability of data-heavy slides across both desktop and mobile devices.

Beyond these specific features, the work also demonstrates how modern CSS capabilities and browser

APIs can be used to simplify responsive layout behaviour. Techniques such as container-aware layout adaptation and dynamic style generation reduce the need for complex JavaScript logic, while improving the flexibility of the presentation environment.

Looking ahead, web browsers continue to evolve into highly capable application platforms. Features that were previously implemented in external frameworks are increasingly supported directly by the web platform itself. As these capabilities expand, browser-based presentation systems can become more powerful, while remaining lightweight and portable.

Overall, this thesis demonstrates that modern web technologies provide a robust foundation for building responsive slide deck systems. By extending and modernising the Rslidy framework, the work contributes a practical implementation that improves responsiveness, print control, and maintainability while preserving the simplicity and flexibility of browser-based presentations.

Appendix A

Slide Viewer Guide

This guide describes how to view and interact with slide decks created using Rslidy v2. It is aimed at end users who consume slide content in a web browser, and does not require prior knowledge of HTML, CSS, or JavaScript. The guide provides an overview of Rslidy's features, describes the user interface and interaction mechanisms, and outlines common usage scenarios. Rslidy v2 supports multiple input modalities, responsive slide layouts, runtime configuration panels, and integrated tools for navigation, printing, and image inspection. An example of a typical Rslidy slide deck can be seen in Figure A.1.

A.1 Installation and Requirements

Rslidy slide decks require no installation beyond a modern web browser. A slide deck consists of static HTML, CSS, and JavaScript files that are opened in the browser like a regular web site. Rslidy v2 is distributed as an ES module. As a result, slide decks must be served over HTTP or HTTPS. Opening a deck directly via `file://` will disable JavaScript execution or prevent loading of external assets in modern browsers. For local viewing, a minimal web server can be started from the slide deck directory, for example using Python:

```
python3 -m http.server 8000
```

The slide deck can then be accessed via:

```
http://localhost:8000
```

Rslidy v2 runs in all modern desktop and mobile browsers. No additional plugins or runtime dependencies are required.

A.2 Overview of Features

Rslidy v2 provides the following features for slide viewers:

- *Responsive Layout*: Slide layouts automatically adapt to different screen sizes and orientations, ensuring that presentations remain readable on both large displays and smaller devices.
- *Light and Dark Modes*: The presentation respects the user's operating system or browser preference, applying light or dark colours as appropriate.
- *Vertical Slide Scrolling*: When slide content exceeds the available viewport height, vertical scrolling allows the remaining content to be accessed without leaving the slide.
- *Multiple Input Modalities*: Presentations can be controlled using keyboard, mouse, and touch input, with optional support for motion-based gestures on compatible devices.

Figure A.1: A typical RSlidy slide deck with slide content, toolbar, and progress bar.

- *Incremental Lists*: List items can be revealed one after another during navigation, helping viewers focus on one point at a time.
- *Progress Bar Navigation*: The progress bar provides a visual indication of the current slide position and allows direct navigation through slide previews.
- *Toolbar*: The toolbar provides access to the primary viewer controls, including for slide navigation, overview, settings, print configuration, and help.
- *Panels*: There are five built-in panels: Slide Overview, Table of Contents, Settings, Print Settings, and Help.
- *Image Viewer*: An integrated image viewer allows images within slides to be opened in a fullscreen overlay with zoom and pan controls.
- *Print and PDF Export*: Advanced print options allow slide output to be configured directly within the viewer, with settings persisted across sessions in the browser's local storage.

A.3 Responsive Slide Layout

Rslidy slide decks adapt dynamically to the available screen space. Unlike fixed-dimension presentation software, slides are not constrained to a single viewport height. When slide content exceeds the visible area, vertical scrolling is enabled automatically. This behaviour ensures that text-heavy slides, large figures, and complex tables remain readable on smaller devices without requiring manual zooming or slide splitting. On larger screens, slides are presented without scrolling whenever possible.

A.4 Light and Dark Modes

Rslidy v2 automatically adapts to the colour scheme selected in the user's operating system or browser. When dark mode is active, Rslidy switches the presentation to a darker colour palette. This applies to

Figure A.2: A typical Rslidy slide deck in dark mode.

slide backgrounds, text, links, the toolbar, progress bar, panels, frames, and SVG-based icons. As a result, slide decks remain readable in both bright and dim viewing environments, without requiring viewers to change settings manually. A dark-mode version of a typical Rslidy slide deck can be seen in Figure A.2.

A.5 Touch Interaction

On touch-enabled devices, slides can be navigated by swiping left or right. Tapping the left or right margin of a slide moves to the previous or next slide respectively. These interactions follow common mobile navigation conventions. Standard browser gestures such as vertical scrolling and pinch-to-zoom remain available for slide content, including images, figures, and long tables. When horizontal scrolling is unavoidable, some devices may require a brief long press before scrolling is activated. The exact behaviour depends on the operating system and browser.

A.6 Keyboard and Mouse Controls

All on-screen controls can be activated using either the mouse or keyboard shortcuts. Slides can be navigated using the arrow keys, Page Up and Page Down keys, or by clicking on the slide margins. Pressing the J key activates the slide number input field, allowing viewers to jump directly to a specific slide by entering its number and confirming with Return. This supports non-linear navigation during discussions or question-and-answer sessions. Slide content font size can be decreased, reset, or increased using the -, R, and + keys. When combined with the Shift key, the same shortcuts adjust the font size of the user interface instead. The Esc key closes any open panel or menu. Extra mouse buttons may also be used for navigation if they are mapped to the browser's history functions.

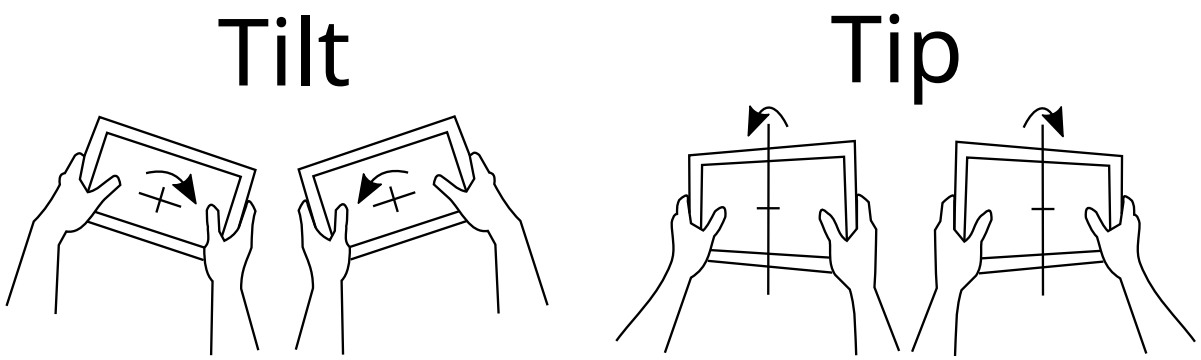


Figure A.3: Tilt and tip gestures can be performed by either tilting or tipping the device towards the left or right side. [Adapted from Hipp [2021b, Figure A.1] and used under the terms of a CC BY 4.0 licence.]

A.7 Motion and Orientation Gestures

On supported devices, motion-based gestures can be enabled for slide navigation. Tilting or tipping the device to the left or right advances slides accordingly. A shake gesture returns to the first slide. An illustration of the tilt and tip gestures can be seen in Figure A.3. To reduce accidental activation, multiple consecutive shakes with sufficient intensity are required. Rslidy adapts automatically to changes in device orientation, unless orientation is locked at the operating system level. Layouts are recomputed dynamically, so that content and user interface remain readable after rotation.

A.8 Incremental Lists

Rslidy v2 provides incremental lists, where list items are revealed one after another instead of appearing all at once. This helps guide attention during a presentation, since each point can be introduced separately. Incremental lists are controlled using the normal slide navigation controls. Pressing the forward navigation key, clicking or tapping to advance, or using the corresponding toolbar button reveals the next hidden list item. After the final list item has been revealed, the next forward action moves to the next slide. To skip incremental reveal steps, hold the Shift key while advancing. This is useful when quickly reviewing a slide deck or moving through slides during a discussion.

A.9 Progress Bar

The progress bar provides a visual overview of the current position within the slide deck. In light mode, previously visited slides are shown in a darker colour, while upcoming slides are displayed in a lighter tone. Hovering over a progress segment reveals a preview thumbnail of the corresponding slide. Selecting a segment navigates directly to that slide without opening additional panels.

A.10 Toolbar

The toolbar is positioned at the bottom of the screen and provides access to Rslidy's primary navigation and configuration features. It can be minimised to maximise available screen space and restored at any time. All toolbar actions are also available via keyboard shortcuts. The toolbar contains the following 12 controls:

- *Slide Overview* (O key): Opens the slide overview panel, which displays thumbnails of all slides and allows quick navigation through the presentation.

- *Hide Toolbar* (T key): Temporarily hides the toolbar to maximise the available screen space, while keeping the progress bar visible.
- *Display All Slides* (A key): Switches from single-slide view to a continuous document view, in which all slides are displayed on a single scrollable page.
- *First Slide* (Home key): Navigates directly to the first slide of the presentation.
- *Previous Slide* (← key): Navigates to the slide preceding the currently displayed slide.
- *Jump to Slide* (J key): Allows direct navigation by entering a specific slide number.
- *Next Slide* (→ key): Navigates to the slide following the currently displayed slide.
- *Last Slide* (End key): Navigates directly to the final slide of the presentation.
- *Print Settings* (P key): Opens the Print Settings panel, which provides controls for configuring printing and PDF export.
- *Settings* (S key): Opens the Settings panel, where interaction techniques and readability options can be adjusted.
- *Help* (H key): Opens the Help panel, which provides an overview of available viewer controls and keyboard shortcuts.
- *Table of Contents* (C key): Opens the Table of Contents panel, which lists slide headings and enables structured navigation through the presentation.

A.11 Panels

Rslidy provides five panels that appear in response to user interaction: Slide Overview, Table of Contents, Settings, Print Settings, and Help. Two of these are illustrated in Figure A.4. Panels can be opened via the toolbar or by using their corresponding keyboard shortcuts. The Esc key closes any open panel.

A.11.1 Slide Overview Panel

The Slide Overview panel, shown on the lefthand side of Figure A.4, displays thumbnails of all slides in the presentation and provides a visual overview of the slide deck. Selecting a thumbnail navigates directly to the corresponding slide. The currently active slide is highlighted to indicate the viewer's position within the presentation. On smaller screens, the width of the panel is constrained to preserve visibility of the slide content.

A.11.2 Table of Contents Panel

The Table of Contents panel, shown on the righthand side of Figure A.4, lists slide headings in document order and provides structured navigation through the presentation. Selecting a heading navigates directly to the corresponding slide. Slides without explicit headings are represented by their slide number.

A.11.3 Settings Panel

The Settings panel, shown in Figure A.5, provides controls for adjusting viewer interaction and readability. The settings are organised into three sections: Enhancements, Interaction, and Font Sizing:

- Enhancements:
  - Show Slide Numbers: Shows or hides slide numbers in the slide deck.

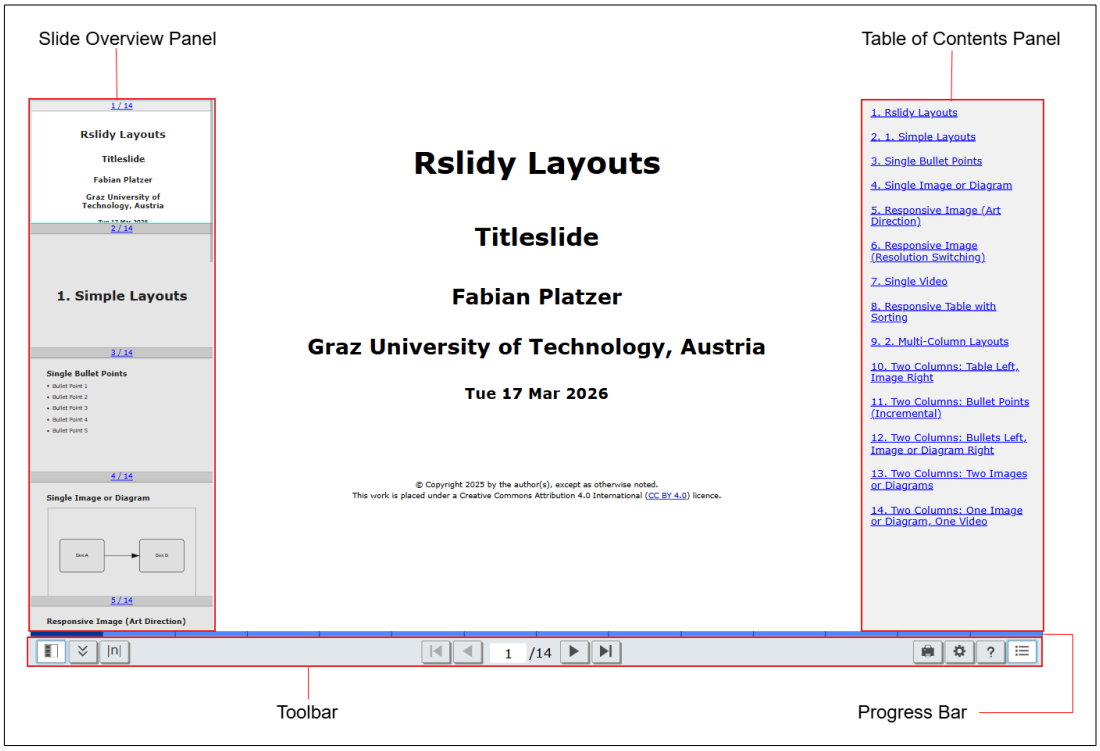


Figure A.4: The Slide Overview and Table of Contents panels.

- Interaction:
  - Tilt: Enables navigation by tilting a supported mobile device.
  - Shake: Enables navigation by shaking a supported mobile device.
  - Space To Advance: Allows viewers to advance through the slide deck using the space key.
  - Margin Tap Nav: Enables navigation by tapping the left or right margin of the slide area.
- Font Sizing:
  - Slide Fonts: Changes the font size used for slide content. Make Slide Fonts Smaller [-] decreases the slide font size, Reset Slide Fonts [R] restores the default value, and Make Slide Fonts Larger [+] increases the slide font size.
  - UI Fonts: Changes the font size used by the Rslidy viewer interface, including the toolbar and menu elements. Make UI Fonts Smaller [Shift -] decreases the interface font size, Reset UI Fonts [Shift R] restores the default value, and Make UI Fonts Larger [Shift +] increases the interface font size.

Settings are persisted under the `rslidy` key in the browser's local storage where permitted. If storage is unavailable, settings remain active only for the current session.

A.11.4 Print Settings Panel

The Print Settings panel, shown in Figure A.6, determines how the slide deck is prepared for printing and PDF export. Print-specific CSS rules are generated at runtime before invoking the browser's print dialogue. The settings are organised into four sections: Print Enhancements, Slides to Print, Page Format, and Print Sizing:

- Print Enhancements:

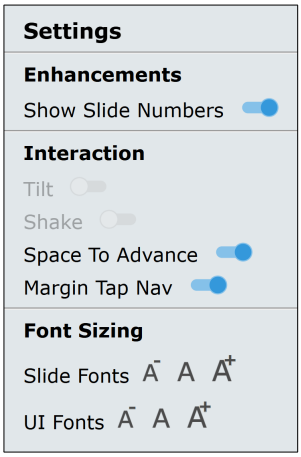


Figure A.5: The Rslidy Settings panel.

  - Show Slide Numbers: Adds the slide number to each printed slide.
  - Show Frame: Draws a frame around each slide, making the slide boundaries clearer on the printed page.
  - Show Link Destinations: Displays the target URLs of links, so that references remain visible in the printed or exported slide deck.

- Slides to Print:
  - All: Prints the complete slide deck.
  - Current: Prints only the slide currently being viewed.
  - Slides: Enables the specification of a custom slide range. The range field supports single slide numbers and intervals, for example `1-5,8,10`.

- Page Format:
  - Page Size: Selects the target paper format, such as A4, A3, Letter, or Legal. This selection is translated into a print-specific `@page` size rule.
  - Orientation: Selects whether the output is generated in portrait or landscape orientation.
  - Font Size: Applies a global font-size multiplier to the printed slides.

- Print Sizing:
  - Actual Size: Keeps the slide at its original dimensions without additional scaling.
  - Fit to Page Width: Scales the slide proportionally, so that it fills the available printable page width.
  - Custom Zoom: Enables a user-defined zoom percentage to enlarge or reduce the printed slide.

Print settings are persisted under the `rslidy-print` key in the browser's local storage. The stored settings are restored automatically when the slide deck is opened again.

A.11.5 Help Panel

The Help panel, shown in Figure A.7, provides an overview of the available viewer controls, keyboard shortcuts, and supported interaction techniques. It serves as a quick reference for users who are unfamiliar with the interface and can be opened either via the toolbar or by pressing the H key.

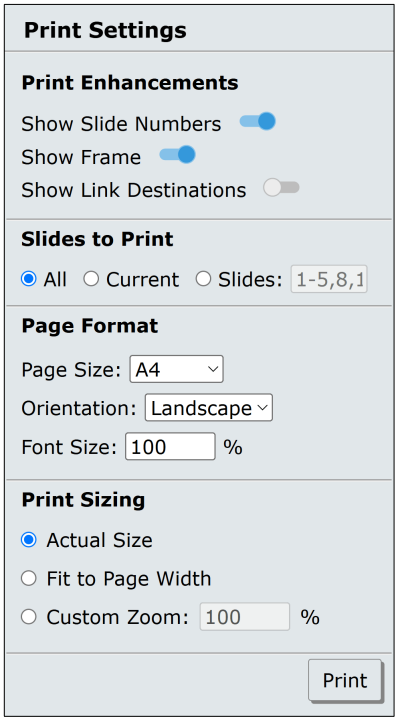
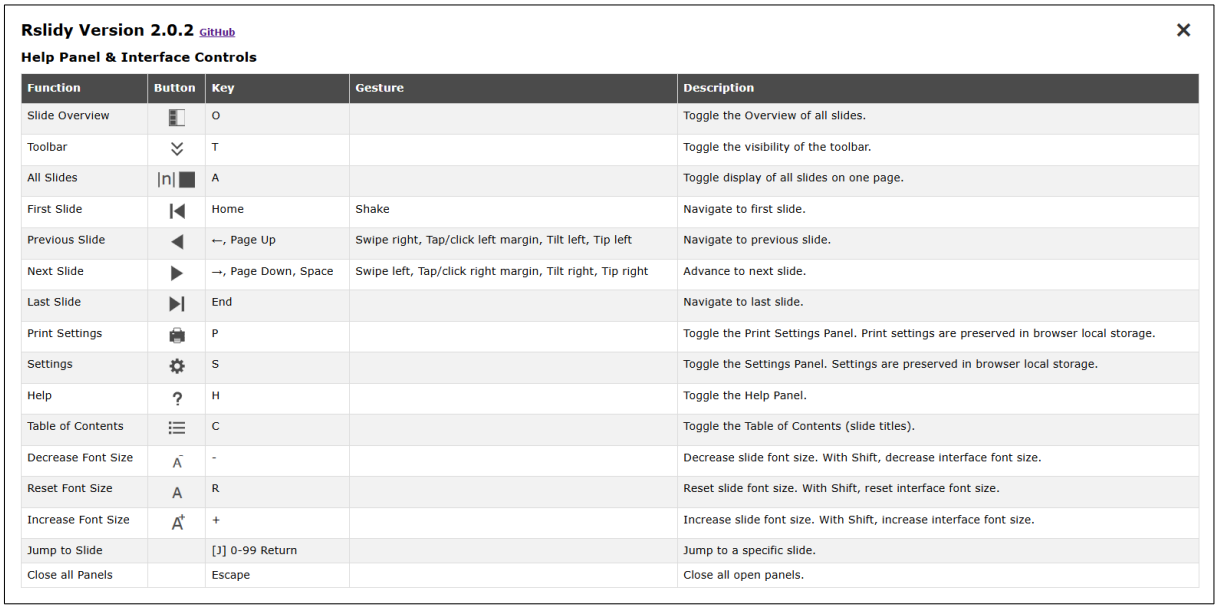


Figure A.6: The Rslidy Print Settings panel.



Function	Button	Key	Gesture	Description
Slide Overview		O		Toggle the Overview of all slides.
Toolbar		T		Toggle the visibility of the toolbar.
All Slides	|n|	A		Toggle display of all slides on one page.
First Slide		Home	Shake	Navigate to first slide.
Previous Slide		←, Page Up	Swipe right, Tap/click left margin, Tilt left, Tip left	Navigate to previous slide.
Next Slide		→, Page Down, Space	Swipe left, Tap/click right margin, Tilt right, Tip right	Advance to next slide.
Last Slide		End		Navigate to last slide.
Print Settings		P		Toggle the Print Settings Panel. Print settings are preserved in browser local storage.
Settings		S		Toggle the Settings Panel. Settings are preserved in browser local storage.
Help		H		Toggle the Help Panel.
Table of Contents		C		Toggle the Table of Contents (slide titles).
Decrease Font Size		-		Decrease slide font size. With Shift, decrease interface font size.
Reset Font Size		R		Reset slide font size. With Shift, reset interface font size.
Increase Font Size		+		Increase slide font size. With Shift, increase interface font size.
Jump to Slide		[J] 0-99 Return		Jump to a specific slide.
Close all Panels		Escape		Close all open panels.

Figure A.7: The Rslidy Help panel.

Figure A.8: The Rslidy Image Viewer overlay provides zoom and pan interactions.

A.12 Image Viewer

Rslidy includes a built-in Image Viewer overlay that is automatically attached to slide images. It is activated by a single left-click and enables zooming and panning using mouse input, keyboard shortcuts, or on-screen controls. The Image Viewer is shown in Figure A.8. Zooming can be done using the mouse wheel, dedicated controls, or keyboard commands. Dragging pans the image, and a reset control restores the default view. The Image Viewer can be closed using the on-screen control, the Esc key, or the browser's back action.

A.13 Common Usage Scenarios

The following are typical usage scenarios for viewing Rslidy slide decks:

- *Jumping to a specific slide*: Press J, enter the slide number, and confirm with Return.
- *Adjusting readability*: Use + and - to change content font size, or hold Shift to adjust the user interface instead.
- *Printing selected slides*: Open the Print Settings panel, select a slide range, choose page format and scaling, and invoke the browser's print dialogue.
- *Inspecting images in detail*: Click an image to open the Image Viewer, zoom using the mouse wheel or controls, and pan by dragging.

A.14 Accessibility

Rslidy is designed to remain usable across a wide range of devices and assistive technologies. All core navigation and configuration features are accessible via keyboard, and interface elements expose appropriate semantic information to screen readers. If JavaScript or advanced CSS features are unavailable,

slides degrade gracefully to a linear, scrollable document view, ensuring that the presentation content remains accessible.

Appendix B

Slide Creator Guide

Rslidy v2 preserves the authoring model of earlier versions, so slide decks remain compatible without structural changes. At the same time, Rslidy v2 introduces selected extensions and refinements that affect integration, responsive media handling, tabular data presentation, theming, printing, and runtime configuration.

Rslidy is distributed as a JavaScript file and a CSS file. Both must be included in the `<head>` of the slide deck. Authors may use either the regular unminified files or the minified variants. The unminified files are preferable during development because they are easier to inspect and debug, while the minified variants reduce file size for deployment.

B.1 Integration and Module-Based Loading

Rslidy v2 is typically loaded as an ES module using a `<script type="module">` element. This aligns integration with modern browser loading semantics and allows additional author scripts to be structured as modules without requiring a bundling step for simple slide decks. A minimal example is shown in Listing B.1. If custom JavaScript is added to a slide deck, it should also be loaded as an ES module to avoid mixing module and classic script execution contexts.

In Rslidy v2, runtime viewer assets are generated through a build workflow based on TypeScript, Gulp, and PNPM. This affects framework development rather than slide authoring directly, but it changes how custom extensions are typically integrated. Authors who customise Rslidy itself should therefore treat the distributed JavaScript and CSS files as build outputs and avoid editing them directly. Project-specific changes should instead be applied in the source files and then rebuilt.

B.2 CSS Variables

Rslidy v2 defines a set of global CSS custom properties (variables) in the file `rslidy/src/css/_variables.css`. These variables provide a central mechanism for adjusting slide dimensions and selected viewer interface properties without changing component-specific rules throughout the stylesheet.

The following variables define the available slide sizes used for layout scaling; default values are given in square brackets [. . .]:

- `--rslidy-large`: Largest slide width [`44rem`].
- `--rslidy-medium`: Default slide width [`40rem`].
- `--rslidy-small`: Reduced slide width [`36rem`].

```
<head>
  <meta charset="UTF-8"/>
  <meta name="viewport"
        content="width=device-width, initial-scale=1"/>

  <title>Rslidy deck</title>

  <link rel="stylesheet" href="rslidy.css"/>
  <script type="module" src="rslidy.js"></script>

  <!-- Deployment variant:
  <link rel="stylesheet" href="rslidy.min.css"/>
  <script type="module" src="rslidy.min.js"></script>
  -->
</head>
```

Listing B.1: Rslidy v2 integration using an ES module script.

- `--rslidy-tiny`: Compact slide width [`32rem`].
- `--rslidy-mini`: Smaller layout width [`28rem`].
- `--rslidy-nano`: Very small width for previews [`24rem`].
- `--rslidy-pico`: Smallest width, used for thumbnails [`20rem`].

The following shared variables are used across multiple viewer components:

- `--rslidy-overview-width`: Width of slides in the Slide Overview panel [`18rem`].
- `--rslidy-overview-width-small-screen`: Overview slide width on narrow screens [`50%`].
- `--rslidy-slide-input-width`: Width of toolbar input fields [`2.60em`].
- `--rslidy-slide-number-font-size`: Font size of the current slide number display [`1rem`].
- `--rslidy-slide-font-size`: Default font size used for normal slide content [`1.5em`].

Toolbar-related variables define colours, dimensions, and interaction states:

- `--rslidy-toolbar-bg-color`: Toolbar background colour [`light-dark(#e1e7ea, #1a1a1a)`].
- `--rslidy-toolbar-button-bg-color`: Button background colour [`light-dark(#e1e7ea, #2a2a2a)`].
- `--rslidy-toolbar-button-color`: Button foreground colour [`light-dark(#4b4b4b, #e0e0e0)`].
- `--rslidy-button-border-color`: Button border colour [`light-dark(#848484, #666)`].
- `--rslidy-button-hover-color`: Button hover state colour [`light-dark(#c1c7ca, #444)`].
- `--rslidy-button-active-color`: Button active state colour [`light-dark(#fff, #000)`].
- `--rslidy-button-disabled-color`: Button disabled colour [`light-dark(#aaa, #555)`].
- `--rslidy-progressbar-bg-color-reached`: Progress bar colour for visited slides [`light-dark(#003399, #6b8cff)`].

```
<section>
  <h1>Incremental List Example</h1>

  <ul class="incremental">
    <li>First point</li>
    <li>Second point</li>
    <li>Third point</li>
  </ul>
</section>
```

Listing B.2: Incremental display of list items using the `incremental` class. Items are revealed one after another during navigation.

- `--rslidy-progressbar-bg-color-unreached`: Progress bar colour for upcoming slides [`light-dark(#4d88ff, #3355aa)`].
- `--rslidy-toolbar-height`: Toolbar height [`3.20em`].
- `--rslidy-progressbar-height`: Progress bar height [`0.50em`].

For the Settings panel, the following variables define slider appearance:

- `--rslidy-slider-fill-off`: Slider fill colour when inactive [`light-dark(#ececec, #333)`].
- `--rslidy-slider-stroke-off`: Slider outline colour when inactive [`light-dark(#bdbdbd, #555)`].
- `--rslidy-slider-fill-on`: Slider fill colour when active [`light-dark(#3498db, #5dade2)`].
- `--rslidy-slider-stroke-on`: Slider outline colour when active [`light-dark(#85c1e9, #2874a6)`].

Since these variables are declared centrally, custom themes can override them to adapt the appearance of the viewer, while preserving the existing layout and component structure. This approach keeps styling changes predictable and avoids inconsistencies caused by overriding individual component rules.

B.3 Incremental Display of List Items

List items can be revealed incrementally instead of appearing all at once by applying the class `incremental` to a <ul> or <ol> element, as shown in Listing B.2. During presentation, items are displayed one after another when navigating forward. To skip incremental steps, the Shift key can be held while advancing slides.

B.4 Slide Transitions

Rslidy v2 supports multiple animation modes for slide transitions. The transition type is defined globally by assigning a class to the <body> element:

```
<body class="unanimated">
```

The following transition modes are available:

- `slidein`: Default sliding transition.

```
<section>
<h1>Image Example</h1>

<figure>
  <img src="images/example.png" alt="Descriptive text."/>
  <figcaption>
    Image captions should remain concise and provide attribution
    where required.
  </figcaption>
</figure>
</section>
```

Listing B.3: Including an image using a standard <img> element.

- zoom: Zoom-based transition between slides.
- fade: Cross-fade transition.
- unanimated: Disables transition animations.

The selected mode is applied consistently to all slides in the deck.

B.5 Images

Images can be included using standard HTML <img> elements. By default, all images are enhanced with the integrated image viewer, which allows zooming and inspection during presentations. The image viewer can be disabled through configuration settings if required. A typical image can be embedded without additional wrapper elements, as shown in Listing B.3.

Authors should ensure that SVG images define intrinsic dimensions, for example by specifying a viewBox inside the SVG, and by specifying at least one of width or height via HTML or CSS. While CSS rules can enforce a minimum size, they do not provide intrinsic aspect ratio information. As a result, they cannot fully compensate for incorrectly defined SVG files, which may lead to unexpected scaling or layout issues. Rslidy also provides optional helper classes for common media layouts. The classes rslidy-images and rslidy-large-images wrap images or videos inside a centred container with a predefined 16:9 aspect ratio:

- rslidy-images: For smaller media elements, limiting the maximum height to 15em.
- rslidy-large-images: For larger media elements, limiting the maximum height to 25em.

These classes are optional layout utilities and are not required for image rendering. They can be applied when a consistent media area or centred presentation is desired.

B.6 Responsive Images

Rslidy supports responsive slide layouts, but the slide creator remains responsible for providing responsive media assets. Two common patterns used for responsive images are art direction and resolution switching:

- *Art Direction*: Selects different image crops depending on the amount of space available. This is useful when scaling a single image would remove essential visual detail. For example, the <picture>

```
<section>
<h1>Responsive Image (Art Direction)</h1>

<figure>
  <picture>
    <source media="(max-width: 40rem)"
            srcset="images/uhrturm-narrow.jpg"/>
    <source media="(max-width: 80rem)"
            srcset="images/uhrturm-medium.jpg"/>
    <img src="images/uhrturm-wide.jpg"
         alt="Clock tower shown with different crops."/>
  </picture>

  <figcaption>
    Different crops preserve important detail on narrow viewports.
  </figcaption>
</figure>
</section>
```

Listing B.4: Art direction using the <picture> element to select different crops.

element can use embedded <source> elements to specify different cropped versions of an image to be used at different viewport widths, as shown in Listing B.4.

- *Resolution Switching*: Allows the browser to select an appropriate image based on the rendered size and device pixel density. The `sizes` attribute describes the expected rendered width under different viewport conditions. An example is shown in Listing B.5.

For both patterns, meaningful alternative text should be provided. If an image is purely decorative, an empty `alt=""` attribute should be used. Authors should also ensure that image proportions remain compatible with the slide layout, because responsive selection improves asset choice, but does not correct unsuitable source imagery. If a predefined media layout is required, the `rslidy-images` and `rslidy-large-images` helper classes can be used to wrap responsive images inside a centred container with a fixed aspect ratio.

B.7 Responsive Tables

Responsive tables are a central extension in Rslidy v2. A table becomes responsive when it is marked with the `rslidy-responsive-table` class. Only such tables receive the responsive layout behaviour and interactive sorting features. On wider screens, the table remains in a conventional tabular layout. When the available container width becomes too small, Rslidy can switch the table into a stacked representation. In this view, the table header is hidden and each body cell is labelled with the corresponding column heading. To support this transformation reliably, header cells should use the `scope="col"` attribute. Rslidy derives cell labels from these headers and injects them automatically. An example of a responsive table following these conventions is shown in Listing B.6. Figure B.1 shows the table in its conventional wide-screen layout. Figure B.2 shows the stacked layout used on narrow screens.

Four utility classes are available to slide authors to improve readability and consistency across table layouts:

- `rslidy-text`: Left-align textual content.

```
<section>
<h1>Responsive Image (Resolution Switching)</h1>

<figure>
  <img
    src="images/uhrturm-800.jpg"
    srcset="
      images/uhrturm-480.jpg 480w,
      images/uhrturm-800.jpg 800w,
      images/uhrturm-1200.jpg 1200w
    "
    sizes="
      (max-width: 40rem) 90vw,
      (max-width: 70rem) 70vw,
      60vw
    "
    alt="Clock tower overlooking Graz."/>
</figure>
</section>
```

Listing B.5: Resolution switching using the `srcset` and `sizes` attributes.

- `rslidy-numeric`: Right-align numeric values.
- `rslidy-symbol`: Centre symbols or icon-like values.
- `rslidy-zebra`: Apply alternating row background styling.
- `rslidy-responsive-table-text-scaling`: Enable automatic font-size scaling for dense tables.

The alignment classes are only required on the corresponding `<th>` elements. Rslidy v2 automatically applies these classes to the related `<td>` elements at runtime, so authors do not need to repeat the same alignment class in every table cell. This keeps the table markup shorter, while preserving consistent alignment across each column.

The base responsive table styling reduces spacing and typography to improve information density. Authors should therefore keep cell content concise and consistent. Long pieces of text, nested markup, or widely varying content types within cells of the same column reduce readability in both wide and stacked layouts.

Sorting is enabled only for responsive tables. On wide screens, sorting is triggered via interaction with column headers. On narrow screens, Rslidy provides a mobile sorting control with a selection dropdown and direction controls. The active sort column and direction remain consistent when switching between desktop and mobile layouts.

If sorting is not desired for a table, it can be disabled while keeping the responsive layout by setting the class `rslidy-disable-sorting` on the `<table>` element. The same table without sorting is shown in Figure B.3. For predictable results, cell values should remain consistent within each column. Mixing numbers, symbols, dates, and free text may lead to ambiguous sort order. Authors can also disable sorting for individual columns by assigning the `rslidy-no-sort` class to the corresponding header cell. This is useful for columns containing controls, decorative symbols, or other content that should remain visible, but should not participate in sorting.

```
<section>
  <h1>Responsive Table with Sorting</h1>
  <table class="rslidy-responsive-table
               rslidy-responsive-table-text-scaling">
    <thead>
    <tr>
      <th scope="col" class="rslidy-text">Smartphone</th>
      <th scope="col" class="rslidy-text">Operating System</th>
      <th scope="col" class="rslidy-numeric">Battery (mAh)</th>
      <th scope="col" class="rslidy-numeric">Display Size ('')</th>
      <th scope="col" class="rslidy-numeric">Storage (GB)</th>
      <th scope="col" class="rslidy-numeric">Price ($)</th>
      <th scope="col" class="rslidy-text">Release Date</th>
    </tr>
    </thead>
    <tbody>
    <tr>
      <td>Samsung Galaxy</td>
      <td>Android</td>
      <td>5000</td>
      <td>6.8</td>
      <td>512</td>
      <td>1199</td>
      <td>2025-03-15</td>
    </tr>
      ...
    <tr>
      <td>Google Pixel</td>
      <td>Android</td>
      <td>4350</td>
      <td>6.4</td>
      <td>256</td>
      <td>799</td>
      <td>2025-01-30</td>
    </tr>
    </tbody>
  </table>
</section>
```

Listing B.6: Responsive table with alignment, scaling, and sorting.

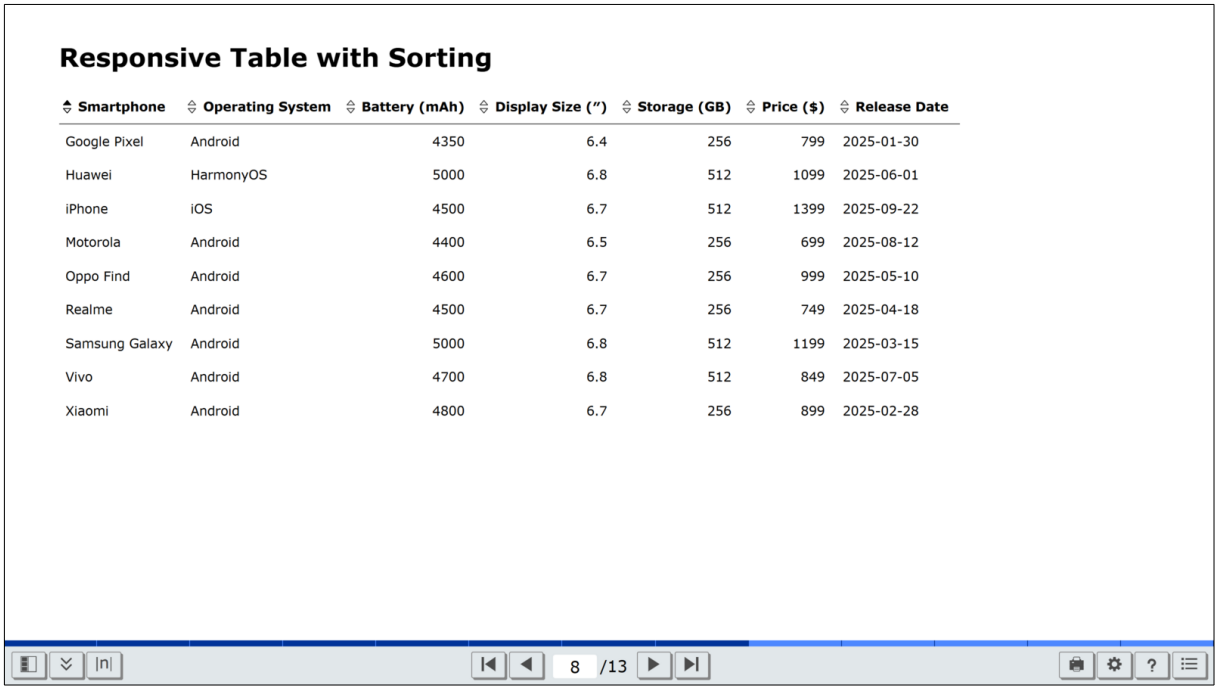


Figure B.1: Responsive table displayed in conventional tabular layout on a wide screen. The corresponding source code is shown in Listing B.6.

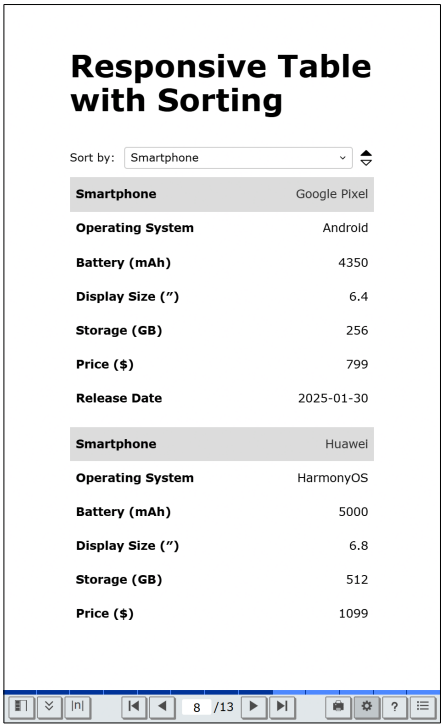


Figure B.2: Responsive table displayed in the stacked layout on a narrow screen. The corresponding source code is shown in Listing B.6.

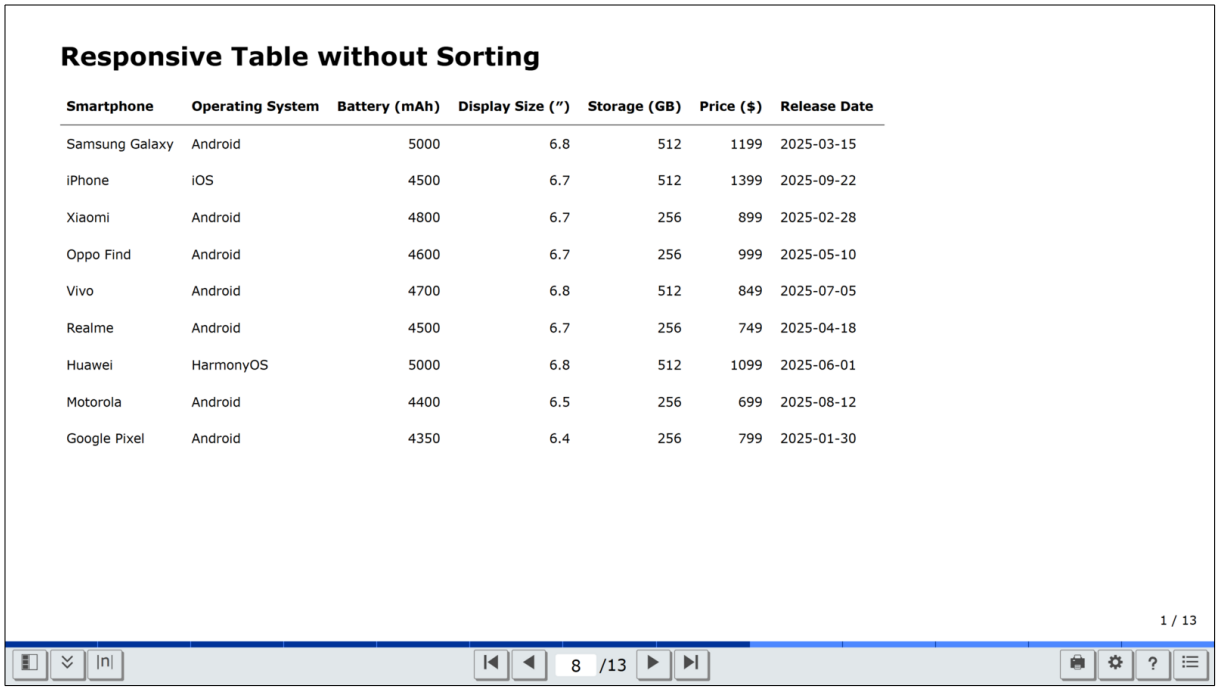


Figure B.3: Responsive table without sorting.

B.8 Interactive and Executable Content

Since Rslidy slide decks run directly in the browser, interactive and executable content can be embedded directly within slides. This allows slide creators to add syntax highlighting, charts, demonstrations, small widgets, or other dynamic elements using JavaScript and third-party libraries. A common example is syntax highlighting for source code using `prism.js` [Verou 2025]. A slide comtaining highlighted source code is shown in Figure B.4. The corresponding source code is shown in Listing B.7.

Executable content should be used carefully. It can improve the presentation experience, but it may not be fully represented in slide thumbnails or printed output. Essential information should therefore remain visible in the static slide content and should not depend only on runtime interaction.

When interactive content depends on external scripts, libraries, or data files, the slide deck must be served through a web server. Modern browsers restrict module loading and fetch operations when a deck is opened directly via a `file://` URL. For local testing, a simple web server can be started in the slide deck directory, for example with the command:

```
python3 -m http.server 8001
```

The deck can then be opened in a browser using the URL `localhost:8001`. All library and data paths should be kept relative to the main HTML file. Absolute paths should be avoided, because broken asset paths are a common cause of interactive content failing in deployed slide decks.

B.9 Helper Classes

Rslidy v2 provides several helper classes for common slide authoring patterns. These classes are optional and define reusable layout behaviour for title slides, section slides, image containers, figures, footers, notes, keyboard labels, and multi-column layouts. They allow slide creators to use consistent slide structures without writing custom CSS for each individual slide. An overview of the most important helper classes is shown in Table B.1.

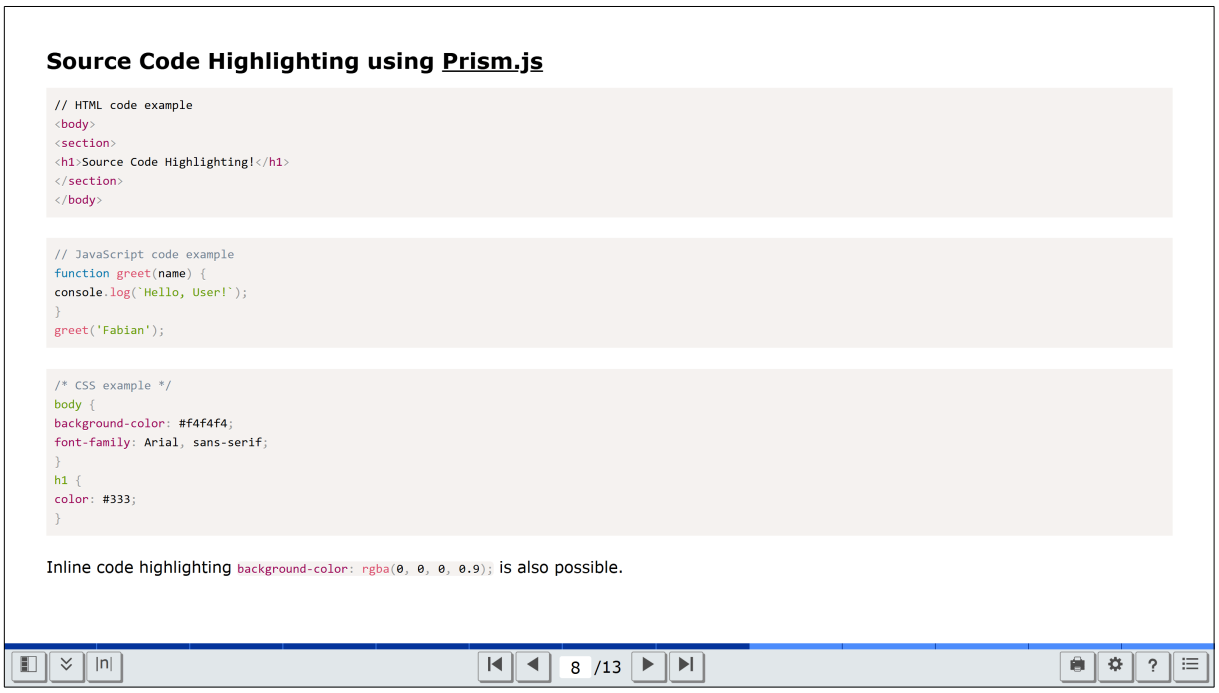


Figure B.4: Code snippets are displayed in the slide deck and syntax highlighted at runtime using `prism.js`. The corresponding source code is shown in Listing B.7.

Class	Description
`rslidy-titleslide`	Centres title-slide content and enlarges the heading hierarchy.
`rslidy-sectionslide`	Centres a section heading both vertically and horizontally.
`rslidy-columns-even`	Arranges content in evenly distributed responsive columns.
`rslidy-column`	Contains one column of stacked content and prevents visual overflow.
`rslidy-images`	Centres smaller images or videos in a 16:9 media area with a limited height.
`rslidy-large-images`	Centres larger images or videos in a 16:9 media area with a larger height limit.
`rslidy-figure-left`	Keeps a figure aligned to the left edge of the slide.
`rslidy-footer`	Pushes small footer content to the bottom right area of a slide.
`rslidy-licence`	Centres compact licence text, usually on the title slide or credits slide.
`rslidy-note`	Displays a smaller note paragraph enclosed in square brackets.
`rslidy-key`	Displays keyboard shortcuts as small bordered key labels.
`rslidy-block`	Displays short code fragments as separated inline blocks.

Table B.1: Helper classes provided by the default Rslidy v2 slide stylesheet.

```
<link rel="stylesheet" href="prism/prism.css"/>
<script src="prism/prism.js"></script>

<section>
<h1>
  Source Code Highlighting using
  <a href="https://prismjs.com/">Prism.js</a>
</h1>
<pre><code class="language-html">// HTML code example
&lt;body&gt;
&lt;section&gt;
&lt;h1&gt;Source Code Highlighting!&lt;/h1&gt;
&lt;/section&gt;
&lt;/body&gt;</code></pre>
<br/>

<pre><code class="language-javascript">// JavaScript code example
function greet(name) {
  console.log(`Hello, ${name}!`);
}
greet('User');</code></pre>
<br/>

<pre><code class="language-css">/* CSS example */
body {
  background-color: #f4f4f4;
  font-family: Arial, sans-serif;
}
h1 {
  color: #333;
}</code></pre>
<br/>

<p>
  Inline code highlighting
  <code class="language-css">
    background-color: rgba(0, 0, 0, 0.9);
  </code>
  is also possible.
</p>
</section>
```

Listing B.7: An Rslidy slide containing source code with syntax highlighting using `Prism.js`.

```
<link rel="stylesheet" href="rslidy.min.css"/>
<link rel="stylesheet" href="themes/tu-graz/theme.css"/>
<script type="module" src="rslidy.min.js"></script>
```

Listing B.8: Theme stylesheets are loaded after the core Rslidy stylesheet.

B.10 Themes and Colour Schemes

Themes are applied by loading an additional stylesheet after the core Rslidy CSS, allowing theme rules to override the defaults. In Rslidy v2, themes typically use CSS custom properties together with automatic light-dark colour adaptation, so the slide deck follows the user's preferred colour scheme without manual toggles. An example is shown in Listing B.8.

Rslidy v2 defines key colours through CSS custom properties prefixed with `--rslidy-`. These variables can be resolved through the CSS `light-dark()` function, allowing one theme definition to contain both light and dark variants. As a result, authors can define a single theme that reacts automatically when the browser or operating system changes its preferred colour scheme.

When authoring custom themes, structural layout rules that support responsive behaviour should not be overridden. Theme-specific styling should focus on colours, typography, and spacing adjustments. In particular, the layout rules for slides, tables, viewer panels, and toolbar elements should be preserved unless a deliberate framework-level customisation is intended.

B.11 Print Configuration

Rslidy v2 extends the original print support with additional controls for paper format, orientation, font scaling, slide selection, and print sizing. These options are configured by the slide viewer at runtime through the Print Settings panel, but slide authors should still prepare content with print output in mind.

Authors should avoid relying on viewport-specific interactions, hover-only states, or content that becomes readable only after manual expansion. Dense slides should be tested with different paper sizes and scaling factors, because the final printed result may vary depending on whether the viewer uses Actual Size, Fit to Page Width, or Custom Zoom. For printable decks, the following authoring practices are recommended:

- *Keep slide layouts stable*: avoid elements that depend on runtime overlap, animation state, or absolute positioning without sufficient margins.
- *Use concise text and clear hierarchy*: heavy text density is more likely to become unreadable when global print font scaling is reduced.
- *Check responsive tables in print preview*: depending on the available width, a table may remain tabular or switch to a stacked layout.

Since Rslidy generates print-specific CSS dynamically at runtime, authors do not need to prepare a separate print stylesheet for ordinary deck usage. However, custom themes or slide-specific CSS styling should be tested with print preview to ensure that they do not conflict with the generated rules.

```
<script>
  window.rslidy.close_menu_on_selection = true;
  window.rslidy.close_navigation_on_selection = true;
  window.rslidy.start_with_toolbar_minimized = true;
  window.rslidy.image_viewer = false;
  window.rslidy.block_slide_text_selection = true;
  window.rslidy.show_slide_dividers = false;
</script>
```

Listing B.9: Global settings are overridden in the <head> of the document.

B.12 Global Settings

Rslidy v2 exposes certain configuration options via the global JavaScript `window.rslidy` object. These global settings must be overridden in the <head> of the document before the slide deck is initialised. Global settings allow slide creators to adjust selected aspects of Rslidy's internal behaviour without modifying the library source code. An example configuration is shown in Listing B.9. The following seven global settings are available in Rslidy v2 for slide creators:

- `close_menu_on_selection`: If set to `true`, menus are closed automatically after a menu item has been selected. The default value is `false`.
- `close_navigation_on_selection`: If set to `true`, the Slide Overview panel and the Table of Contents panel are closed after a slide has been selected. The default value is `false`.
- `start_with_toolbar_minimized`: If set to `true`, Rslidy starts with the toolbar minimised. The default value is `false`.
- `image_viewer`: If set to `false`, disables the image viewer component. The default value is `true`.
- `block_slide_text_selection`: If set to `true`, text selection inside slides is blocked during normal interaction. This can be useful for presentation-focused decks.
- `show_slide_dividers`: If set to `true`, visual dividers between slides are shown in the progress bar. The default value is `true`.
- `close_menu_on_unfcous`: If set to `true`, menus are closed when they lose focus. The default value is `true`.

Appendix C

Developer Guide

This Developer Guide describes the development setup, build pipeline, tooling, and project structure relevant for contributing to the codebase or for adapting Rslidy v2 to specialised use cases. It is aimed at developers who want to build, inspect, modify, or extend Rslidy v2.

C.1 Development Setup

Rslidy v2 is developed using Node.js. The minimum required version is 16. Dependency management and task execution are handled using PNPM, which can be installed globally via NPM:

```
npm install -g pnpm
```

To build Rslidy v2 from source, the repository must first be cloned. Cloning via SSH is recommended and requires a configured GitHub SSH key:

```
git clone git@github.com:tugraz-isds/rslidy.git
```

Alternatively, the repository can be cloned using HTTPS:

```
git clone https://github.com/tugraz-isds/rslidy.git
```

After cloning, the project dependencies must be installed:

```
cd rslidy
pnpm install
```

To build Rslidy, run the command:

```
pnpm exec gulp build
```

The `build/` directory then contains the compiled JavaScript and CSS files which comprise Rslidy v2, as well as some example slide decks.

For interactive development, a local development server with automatic rebuild and reload can be started using:

```
pnpm exec gulp watch
```

A specific slide deck can be watched using the `-slide` option, which must refer to a directory within `examples/` or `tests/`:

```
pnpm exec gulp watch --slide examples/layouts
```

or

```
pnpm exec gulp watch --slide tests/stress-test
```

When modifying or adding icons, the SVG files in `src/icons/` must be processed explicitly, by running the command:

```
pnpm exec gulp icons
```

This updates the file `src/ts/icon-definitions.ts` used by the TypeScript codebase.

C.2 Build Pipeline and Tooling

Rslidy v2 uses Gulp as a task runner and Webpack as a module bundler. Together, these tools specify a reproducible build pipeline that covers TypeScript transpilation, asset processing, bundling, optimisation, minification, and local development support. All build logic is defined in the `gulpfile.js` file. Gulp tasks should be executed using a command like:

```
pnpm exec gulp <task>
```

This ensures that the project's local Gulp installation and its dependencies are used consistently across development environments.

C.3 Gulp Tasks

While the composite Gulp tasks `build` and `watch` are sufficient for most development scenarios, Rslidy defines a number of smaller tasks that implement individual pipeline steps. These tasks can be executed independently when debugging or extending the build process. The following Gulp tasks are available in the Rslidy build pipeline:

- `clean`: Removes build artefacts, including transpiled TypeScript output and the compiled `build/` directory.
- `transpile`: Transpiles all TypeScript source files into JavaScript and generates corresponding declaration files.
- `webpack`: Bundles the transpiled JavaScript into distributable builds in the ESM, CommonJS, and UMD formats.
- `css`: Concatenates all CSS source files into a single stylesheet.
- `icons`: Optimises SVG files in `src/icons/` and generates the `icon-definitions.ts` file used by the TypeScript codebase.
- `html`: Copies example and test HTML files into the `build/` directory.
- `minifyjs`: Produces minified JavaScript bundles for all module formats.
- `minifycss`: Generates a minified version of the main CSS file.
- `compress`: Produces compressed variants of JavaScript bundles to assess distribution size.
- `copy`: Copies the default ESM module build and CSS files into all example and test directories.
- `build`: Executes the full build pipeline and produces all distributable artefacts.
- `watch`: Starts a local development server using BrowserSync and automatically rebuilds and reloads the browser when source files change. A specific slide deck can be served using the `--slide` (or `-s`) option.

- `updateVersionStrings`: Updates version identifiers in the `src/` folder by replacing occurrences of `Rslidy Version X.Y.Z` and `__VERSION__` with the version defined in `package.json`.

These tasks are composed explicitly in the build pipeline, which makes it straightforward to insert additional processing steps or to adapt existing ones.

C.4 Build Output

After a successful build, distributable artefacts are collected in the `build/library/` directory. Rslidy v2 produces three module formats to support different integration scenarios:

- `esm/`: ES Module (ESM) builds. The default recommended when using Rslidy v2.
- `cjs/`: CommonJS (CJS) builds for Node.js-based tooling.
- `umd/`: Universal Module Definition (UMD) builds for legacy integration scenarios.

In addition, the compiled stylesheets `rslidy.css` and `rslidy.min.css` are placed in the `library/` directory too. The minified ESM build is automatically copied into all example and test folders, as it represents the recommended default for Rslidy v2 slide decks.

Since ESM modules must be served over HTTP(S), the built example slide decks cannot be opened directly from the file system. A local web server can be started using Python:

```
python3 -m http.server 8001
```

and the examples can then be accessed in a web browser with the URL `localhost:8001`.

C.5 Development Dependencies

Rslidy v2 does not require any external runtime dependencies. The library is distributed as a single prebuilt JavaScript file `rslidy.js` (or `rslidy.min.js`) and a single CSS file `rslidy.css` (or `rslidy.min.css`).

All additional packages are required exclusively for development, building, and testing. They are defined as development dependencies in the `package.json` file. The development dependencies can be grouped by their primary purpose:

- *Node.js, PNPM, and Gulp*: Provide the execution environment and task runner used to define and orchestrate the build pipeline.
- *TypeScript and related tooling*: Used to transpile the TypeScript source code into JavaScript and to generate declaration files for tooling support.
- *Webpack and integration helpers*: Bundle transpiled modules into distributable builds for multiple module formats.
- *BrowserSync*: Provides a local development server with automatic reloading during the `watch` task.
- *File system and utility tools*: Support cleaning build directories, concatenating assets, renaming outputs, and applying textual transformations.
- *Minification and compression tools*: Generate minified and compressed JavaScript and CSS files for assessing distribution size.
- *Command-line argument handling*: Enables Gulp tasks to accept options such as the `--slide` parameter for serving a specific deck during development.

This separation between runtime and development dependencies ensures that slide decks using Rslidy remain lightweight, while developers retain full control over the build and extension process.

Bibliography

Adobe [2026]. *Illustrator*. 22 Jun 2026. https://adobe.com/products/illustrator/ (cited on page 5).

Andrews, Keith [2021]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Technical report. Graz University of Technology, 10 Nov 2021. https://ftp.isds.tugraz.at/pub/keith/thesis (cited on page xiii).

Apple [2026]. *Apple Keynote*. 22 Jun 2026. https://apple.com/in/keynote/ (cited on pages 1, 11).

Bang, Ole Petter [2023]. *Remark*. 27 Jun 2023. https://github.com/gnab/remark (cited on pages xiii, 13, 16).

Bauer, Gerald [2017]. *Slide Show (S9)*. Date refers to the latest default-branch commit of the slideshow-s9.github.io repository. 28 Nov 2017. https://slideshow-s9.github.io/ (cited on page 13).

Baumgartner, Stefan [2023]. *TypeScript Cookbook*. O'Reilly, 12 Sep 2023. 419 pages. ISBN 1098136659 (cited on page 6).

Bierman, Gavin, Martín Abadi and Mads Torgersen [2014]. *Understanding TypeScript*. Proc. 28th European Conference on Object-Oriented Programming (ECOOP 2014) (Uppsala, Sweden). LNCS 8586. Springer, 28 Jul 2014, pages 257–281. doi:10.1007/978-3-662-44202-9_11. https://gavinbierman.github.io/assets/pdf/ecoop2014.pdf (cited on page 6).

Bublitz, Blaine and Eric Schoffstall [2026]. *Gulp*. 09 Feb 2026. https://github.com/gulpjs/gulp (cited on pages 7, 30).

CommonJS [2026]. *CommonJS Modules*. 22 Jun 2026. https://nodejs.org/api/modules.html (cited on page 7).

Cone, Matt [2026]. *Markdown Guide*. 22 Jun 2026. https://markdownguide.org/ (cited on page 13).

Dalgleish, Mark [2020]. *Bespoke.js*. 08 Sep 2020. https://github.com/bespokejs/bespoke (cited on pages xiii, 30).

ECMA [2015]. *ECMAScript 2015 Language Specification*. Ecma International TC39, Jun 2015. https://262.ecma-international.org/6.0/ (cited on page 6).

ECMA [2026]. *ECMAScript 2026 Language Specification*. Ecma International TC39, 22 Jun 2026. https://tc39.es/ecma262/ (cited on pages 6–7).

El Hattab, Hakim [2026]. *reveal.js*. 21 May 2026. https://github.com/hakimel/reveal.js (cited on pages xiii, 26–27).

El Hattab, Hakim and Owen Bossola [2026]. *slides*. 22 Jun 2026. https://slides.com/ (cited on page 27).

Flanagan, David [2020]. *JavaScript: The Definitive Guide*. 7th Edition. O'Reilly, 23 Jun 2020. 706 pages. ISBN 1491952024 (cited on pages 1, 6).

Fu, Anthony [2026a]. *Slidev*. 03 Jun 2026. https://github.com/slidevjs/slidev (cited on pages xiii, 19).

Fu, Anthony [2026b]. *Slidev: Getting Started*. Date refers to the latest default-branch commit of the Slidev repository. 03 Jun 2026. https://sli.dev/guide (cited on page 21).

Fu, Anthony [2026c]. *Slidev: Try It Online*. Date refers to the latest default-branch commit of the Slidev repository. 03 Jun 2026. https://sli.dev/new (cited on page 21).

Garfield, James [2016]. *Asynchronous Module Definition (AMD)*. 09 Feb 2016. https://github.com/amdjs/amdjs-api (cited on page 7).

Google [2026a]. *Angular*. 19 Jun 2026. https://angular.dev/ (cited on page 6).

Google [2026b]. *Google Slides*. 22 Jun 2026. https://workspace.google.com/products/slides/ (cited on pages xiii, 36).

Govett, Devon [2026]. *Parcel: Zero Configuration Build Tool*. 02 Feb 2026. https://parceljs.org/ (cited on page 9).

Gruber, John [2004]. *Markdown*. 17 Dec 2004. https://daringfireball.net/projects/markdown/ (cited on page 13).

Halácsy, Péter, Peter Arvai and Adam Somlai-Fischer [2026]. *Prezi*. 22 Jun 2026. https://prezi.com/ (cited on pages xiii, 28, 38).

Hattori, Yuki [2026]. *Marp*. 01 May 2026. https://github.com/marp-team/marp (cited on pages xiii, 15).

Hipp, Patrick [2019]. *Slide Decks in HTML*. Survey Paper. Graz University of Technology, 25 Jan 2019. https://ftp.isds.tugraz.at/pub/papers/hipp-2019-01-25-survey-slide-decks.pdf (cited on pages 11, 13).

Hipp, Patrick [2021a]. *Rslidy Version 1.0.1*. 29 Jan 2021. https://github.com/tugraz-isds/rslidy/releases/tag/v1.0.1 (cited on page 49).

Hipp, Patrick [2021b]. *Rslidy: Lightweight, Accessible, and Responsive HTML5 Slide Decks*. Master's Thesis. Graz University of Technology, 11 Sep 2021. https://ftp.isds.tugraz.at/pub/theses/phipp-2021-msc.pdf (cited on pages 1, 46, 49, 51, 78).

Hiroto, Yuta [2021]. *Fusuma*. Repository archived on 2024-12-05; read-only. 01 Dec 2021. https://github.com/hiroppy/fusuma (cited on page 19).

Inkscape [2026]. *Inkscape*. 22 Jun 2026. https://inkscape.org/ (cited on page 5).

Ludvigsen, Holger and Espen H. Halvorsen [2016]. *Slidifier*. 27 Jan 2016. https://github.com/holgerl/Slidifier (cited on page 13).

MacFarlane, John, Martin Woodward and Jeff Atwood [2026]. *CommonMark*. 27 Apr 2026. https://github.com/commonmark/commonmark-spec (cited on page 15).

Makeev, Vadim [2026]. *Shower*. 15 Jun 2026. https://github.com/shower/shower (cited on pages xiii, 24).

Marcotte, Ethan [2010]. *Responsive Web Design*. 25 May 2010. https://alistapart.com/article/responsive-web-design/ (cited on page 9).

Marcotte, Ethan [2015]. *Responsive Web Design*. A Book Apart, 12 Jun 2015. https://abookapart.com/products/responsive-web-design (cited on page 9).

MDN [2025a]. *CSS Values and Units*. 19 Dec 2025. https://developer.mozilla.org/docs/Learn_web_development/Core/Styling_basics/Values_and_units (cited on page 10).

MDN [2025b]. *IndexedDB API*. 03 Apr 2025. https://developer.mozilla.org/docs/Web/API/IndexedDB_API (cited on page 6).

MDN [2025c]. *Responsive web design*. 19 Dec 2025. https://developer.mozilla.org/docs/Learn/CSS/CSS_layout/Responsive_Design (cited on page 9).

MDN [2025d]. *SVG in HTML introduction*. 13 Aug 2025. https://developer.mozilla.org/docs/Web/SVG/Guides/SVG_in_HTML (cited on page 4).

MDN [2025e]. *SVG: Scalable Vector Graphics*. 30 Oct 2025. https://developer.mozilla.org/docs/Web/SVG (cited on page 4).

MDN [2025f]. *The Web Standards Model*. 19 Nov 2025. https://developer.mozilla.org/docs/Learn_web_development/Getting_started/Web_standards/The_web_standards_model (cited on page 57).

MDN [2025g]. *Web APIs*. 29 Jul 2025. https://developer.mozilla.org/docs/Web/API (cited on page 6).

MDN [2026a]. *<details>: The Details disclosure element*. 24 Apr 2026. https://developer.mozilla.org/docs/Web/HTML/Element/details (cited on page 3).

MDN [2026b]. *<picture>: The Picture element*. 24 Apr 2026. https://developer.mozilla.org/docs/Web/HTML/Element/picture (cited on page 3).

MDN [2026c]. *CSS Container Queries*. 08 Jun 2026. https://developer.mozilla.org/docs/Web/CSS/CSS_Container_Queries (cited on page 5).

MDN [2026d]. *Fetch API*. 08 Jun 2026. https://developer.mozilla.org/docs/Web/API/Fetch_API (cited on page 6).

MDN [2026e]. *JavaScript modules*. 04 Apr 2026. https://developer.mozilla.org/docs/Web/JavaScript/Guide/Modules (cited on page 7).

MDN [2026f]. *Using container size and style queries*. 08 Jun 2026. https://developer.mozilla.org/docs/Web/CSS/Guides/Containment/Container_size_and_style_queries (cited on page 5).

MDN [2026g]. *Web forms*. 17 Mar 2026. https://developer.mozilla.org/docs/Learn/Forms (cited on page 3).

MDX [2026]. *MDX*. Compositor and Vercel, 17 Jun 2026. https://github.com/mdx-js/mdx (cited on page 19).

Meta [2026a]. *JSX*. 22 Jun 2026. https://react.github.io/jsx (cited on page 19).

Meta [2026b]. *React*. 18 Jun 2026. https://react.dev/ (cited on page 6).

Meyer, Eric A. and Estelle Weyl [2023]. *CSS: The Definitive Guide: Web Layout and Presentation*. 5th Edition. O'Reilly, 04 Jul 2023. 1126 pages. ISBN 1098117611 (cited on page 9).

Microsoft [2026a]. *Microsoft PowerPoint*. 22 Jun 2026. https://powerpoint.cloud.microsoft/ (cited on pages 1, 11).

Microsoft [2026b]. *Microsoft Sway*. 22 Jun 2026. https://sway.cloud.microsoft/ (cited on pages xiii, 43).

Nakajima, Pat and Dan Croak [2012]. *Slidedown*. 16 Feb 2012. https://github.com/nakajima/slidedown (cited on page 13).

Netlify [2026]. *Netlify*. 22 Jun 2026. https://netlify.com/ (cited on page 24).

Node [2026]. *Node.js*. 20 Jun 2026. https://nodejs.org/ (cited on pages 6, 13, 19).

NPM [2026a]. *npm*. 22 Jun 2026. https://npmjs.com/ (cited on pages 2, 23, 52–53).

NPM [2026b]. *npm bespoke plugins*. 22 Jun 2026. https://npmjs.com/search?q=keywords:bespoke-plugin (cited on page 30).

OJSF [2026]. *Webpack*. OpenJS Foundation, 21 Jun 2026. https://webpack.js.org/ (cited on pages 9–10).

Platzer, Fabian [2024]. *Web-Based Slide Decks*. Survey Paper. Graz University of Technology, 09 Dec 2024. https://ftp.isds.tugraz.at/pub/surveys/platzer-2024-12-09-survey-web-slide-decks.pdf (cited on page 11).

Platzer, Fabian [2026]. *Rslidy Version 2.1.0*. 30 Jun 2026. https://github.com/tugraz-isds/rslidy/releases/tag/v2.1.0 (cited on page 53).

PNPM [2026]. *pnpm*. 21 Jun 2026. https://pnpm.io/ (cited on page 53).

Raggett, Dave [2005]. *HTML Slidy: Slide Shows in XHTML*. Mar 2005. https://w3.org/2005/03/slideshow.html (cited on page 11).

Raggett, Dave [2006]. *Slidy2: Slide Shows in XHTML*. 2006. https://w3.org/Talks/Tools/Slidy2 (cited on pages 11–12, 46, 49).

Rollup [2026]. *Rollup: The JavaScript Module Bundler*. Rollup Contributors, 19 Jun 2026. https://rollupjs.org/ (cited on page 9).

Rslidy [2026]. *Rslidy: Lightweight, Accessible, and Responsive HTML5 Slide Decks*. 14 Jun 2026. https://github.com/tugraz-isds/rslidy (cited on page 46).

Szopka, Bartek and Henrik Ingo [2025]. *impress.js*. 25 Sep 2025. https://github.com/impress/impress.js (cited on pages xiii, 28).

Taei, Payman [2026a]. *Visme*. 22 Jun 2026. https://visme.co/ (cited on pages xiii, 40).

Taei, Payman [2026b]. *Visme Pricing*. 22 Jun 2026. https://visme.co/pricing/ (cited on page 40).

Troughton, Caleb [2016]. *Deck.js*. 04 May 2016. https://github.com/imakewebthings/deck.js (cited on pages xiii, 24).

UMD [2024]. *Universal Module Definition (UMD)*. UMD Contributors, 26 Nov 2024. https://github.com/umdjs/umd (cited on page 7).

Verou, Lea [2025]. *PrismJS*. 21 May 2025. https://github.com/PrismJS/prism (cited on page 93).

Verou, Lea [2026]. *Inspire.js*. 08 Jun 2026. https://github.com/inspire-js/inspire.js (cited on pages xiii, 32).

Vite [2026]. *Vite: Next-Generation Frontend Tooling*. VoidZero and Vite contributors, 20 Jun 2026. https://vite.dev/ (cited on page 9).

W3C [2011]. *Scalable Vector Graphics (SVG) 1.1*. W3C Recommendation. 16 Aug 2011. https://w3.org/TR/SVG11/ (cited on page 4).

W3C [2014]. World Wide Web Consortium (W3C), 28 Oct 2014. https://w3.org/TR/2014/REC-html5-20141028/ (cited on page 3).

W3C [2018]. *Scalable Vector Graphics (SVG) 2*. W3C Candidate Recommendation. 04 Oct 2018. https://w3.org/TR/SVG2/ (cited on page 4).

W3C [2026]. *HTML DOM (Document Object Model)*. 22 Jun 2026. https://w3schools.com/js/js_htmldom.asp (cited on page 4).

Warström, Johnny, Niklas Ingvar, Kristoffer Renholm and Thomas Toti [2026]. *Mentimeter*. 22 Jun 2026. https://mentimeter.com/ (cited on pages xiii, 41).

web.dev [2026]. *Baseline*. 22 Jun 2026. https://web.dev/baseline (cited on pages 3, 5).

WHATWG [2026]. *HTML Living Standard*. 22 Jun 2026. https://html.spec.whatwg.org/ (cited on page 3).

Wirfs-Brock, Allen and Brendan Eich [2020]. *JavaScript: The First 20 Years*. Proceedings of the ACM on Programming Languages 4.HOPL (12 Jun 2020). doi:10.1145/3386327. https://zenodo.org/records/4960086 (cited on page 6).

WPT [2026]. *Interop 2025 Dashboard*. web-platform-tests, 22 Jun 2026. https://wpt.fyi/interop-2025 (cited on pages 3, 5).

Yarn [2026]. *Yarn*. 15 Jun 2026. https://github.com/yarnpkg/berry (cited on page 53).

You, Evan [2026]. *Vue.js*. 17 Jun 2026. https://github.com/vuejs/core (cited on pages 6, 19).

Zoho [2026]. *Zoho Show*. 22 Jun 2026. https://zoho.com/show/ (cited on pages xiii, 37).