OPEN SOURCE ASSESSMENT FRAMEWORK

PIE: Portable Interactions & Elements

A modern, production-ready framework for interactive assessment item rendering. Built on web standards. Open source.

TECHNICAL PRIMER · 2026

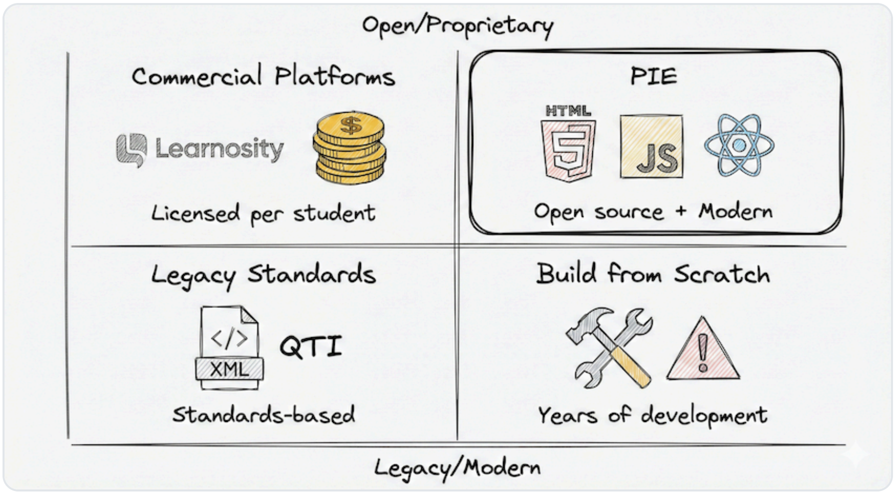
CONTENTS

Educational assessment has evolved far beyond multiple-choice bubble sheets. Today's assessments demand interactive question types—graphing functions, dragging items into categories, annotating images, constructing mathematical expressions—all delivered through web browsers to millions of students. These assessments must be accessible to students with diverse needs, compliant with rigorous standards like WCAG 2.2, and flexible enough to support various pedagogical approaches.

Organizations building assessment platforms face a difficult choice. Commercial platforms like Learnosity provide comprehensive solutions but come with significant per-student licensing costs and limited customization. Standards like QTI provide content interoperability, but QTI is a data transport standard—it defines how items are exchanged, not how they look or how they're authored. In practice, the same QTI item can render very differently across platforms (especially for technology-enhanced items), and QTI says nothing about authoring interfaces, which matters greatly for content teams. Building from scratch provides control but demands years of development to cover a wide range of content with production quality.

PIE (Portable Interactions and Elements) offers a different path: a modern, open-source framework for assessment item rendering that combines the interactive richness of commercial platforms with the flexibility of open standards. Built on web standards and distributed as framework-agnostic Web Components, PIE provides production-ready question types, authoring interfaces, and delivery players while letting you maintain complete control over your backend infrastructure.

PIE isn't theoretical. Renaissance Learning, one of the largest educational technology companies globally, has standardized on PIE as their assessment framework, serving hundreds of millions of student interactions annually across 40% of U.S. schools and in various other countries. This primer explains what PIE is, how it works architecturally, and why organizations building assessment and item bank solutions should consider it for their architectures.

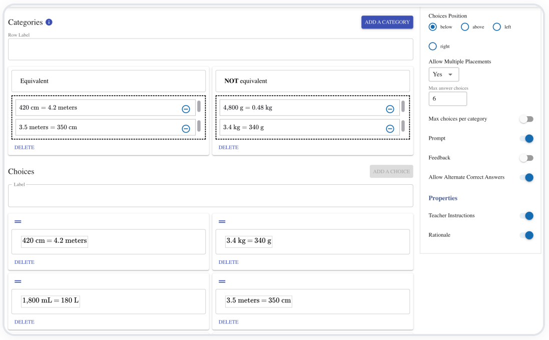


PIE bridges the gap between proprietary platforms and building from scratch

SECTION 01

What is PIE?

PIE is an open-source frontend framework for rendering interactive assessment items. Unlike commercial platforms that bundle frontend, backend, and hosting into a single service, PIE focuses exclusively on what appears in the browser: the question types, authoring interfaces, and delivery players. You bring your own backend—your content management system, your item bank, your assessment administration platform, your authentication system, your scoring engine, your infrastructure. These backend systems can be unified or separate (authoring CMS, production delivery system) depending on your architectural needs.



Equivalent: 420 cm = 4.2 meters; 3.5 meters = 350 cm

NOT equivalent: 4,800 g = 0.48 kg; 3.4 kg = 340 g

Choices: 420 cm = 4.2 meters; 3.4 kg = 340 g; 1,800 mL = 180 L; 3.5 meters = 350 cm

Choices Position: [✓ below] above left right

Allow Multiple Placements: Yes

Max answer choices: 6

Max choices per category: off

Prompt: on

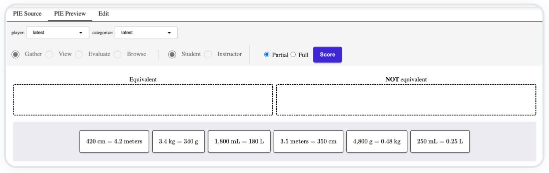
Feedback: off

Allow Alternate Correct Answers: on

Teacher Instructions: on

Rationale: on

Categorize — Authoring



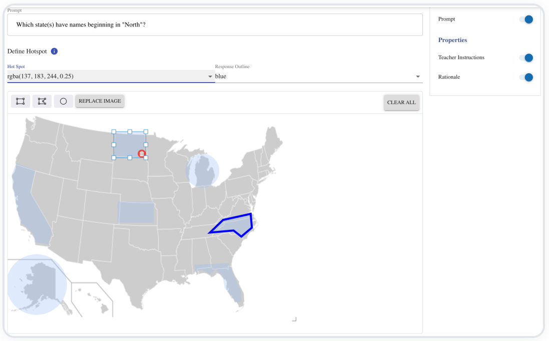
player: latest categorize: latest

[✓ Gather] View Evaluate Browse | [✓ Student] Instructor | [✓ Partial] Full

Equivalent | NOT equivalent

420 cm = 4.2 meters | 3.4 kg = 340 g | 1,800 mL = 180 L | 3.5 meters = 350 cm | 4,800 g = 0.48 kg | 250 mL = 0.25 L

Categorize — Student View



Prompt: Which state(s) have names beginning in "North"?

Hot Spot: rgba(137, 183, 244, 0.25)

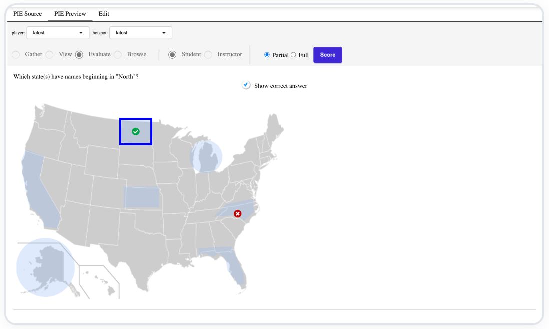
Response Outline: blue

Prompt: on

Teacher Instructions: on

Rationale: on

Hotspot — Authoring



player: latest hotspot: latest

Gather View [✓ Evaluate] Browse | [✓ Student] Instructor | [✓ Partial] Full

Which state(s) have names beginning in "North"?

✓ Show correct answer

Hotspot — Evaluation

Core Architecture

PIE's architecture separates concerns cleanly:

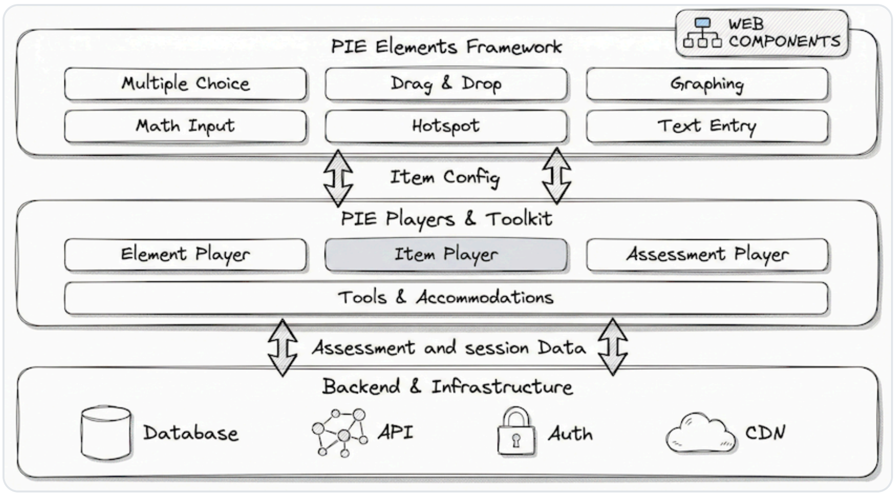
PIE PROVIDES

- A library of 20+ production-ready question types (multiple choice, graphing, drag-and-drop, constructed response, etc.)
- WYSIWYG authoring interfaces for educators to create content without writing code
- A layered player hierarchy—item players render individual questions, section players compose items with passages and tools into page-level views, and assessment players orchestrate navigation across sections—each usable independently depending on how much of the stack you need
- Built-in accessibility features aiming to fully support WCAG 2.2 Level AA standards
- An assessment toolkit with calculator, text-to-speech, accommodation management, and other services, coordinated through a centralized ToolkitCoordinator
- QTI-inspired models for sections and assessments, making PIE content naturally mappable to QTI 3.0 structures

YOU PROVIDE

- Item storage and retrieval (your CMS or item bank)
- User authentication and authorization
- Session management and persistence
- Reporting
- Assessment administration workflows

This separation means PIE can integrate with any backend technology you wish.



High-level architecture — PIE handles frontend rendering; you handle the backend

Key Differentiators

WYSIWYG Authoring

Graphical authoring interfaces with live previewing. Authors configure questions through forms and visual editors, seeing exactly how students will experience the content.

Modern Web Standards

Packaged as HTML5 Web Components (custom elements), the browser-native standard for reusable UI components. Works with React, Vue, Angular, Svelte, or vanilla JavaScript.

Framework-Agnostic

Because PIE uses Web Components, you're not locked into a particular frontend technology. Migrate frameworks without modifying PIE integrations.

Production-Ready

Powers some of the highest-volume assessment platforms in education, processing hundreds of millions of student interactions yearly with enterprise-grade reliability.

Accessibility First

Every element includes keyboard navigation, screen reader support, ARIA labels, and high-contrast theme compatibility — architected in from the beginning.

Standards Alignment

PIE-QTI provides production-ready QTI 2.x and 3.0 players with a version-agnostic architecture. Bidirectional PIE-to-QTI transforms are under active development.

Development and Testing

PIE can be tested and developed without backend infrastructure. The framework includes standalone demos and an interactive playground where you can configure items and see them render immediately. This accelerates development cycles and enables frontend teams to work independently. When you're ready for production, PIE integrates with your backend through straightforward JSON APIs and event handlers.

Open Source Software

PIE is licensed with permissive open source software licensing (ISC/MIT) and is being developed by a sizable team of full-time developers. At Renaissance, PIE's main sponsor, we aim to accelerate learning for all children and adults of all ability levels and ethnic and social backgrounds, worldwide, and we hope that contributing to PIE will contribute to that.

SECTION 02

PIE Elements: The Question Type Framework

At the foundation of PIE are **elements**—reusable, interactive question components that represent different assessment item types. Each element encapsulates the logic, user interface, and accessibility features needed for both authoring content and delivering it to students.

Multiple Interface Design

Every PIE element provides at least:

Delivery Interface: The interactive UI students see and interact with. This mode adapts based on context:

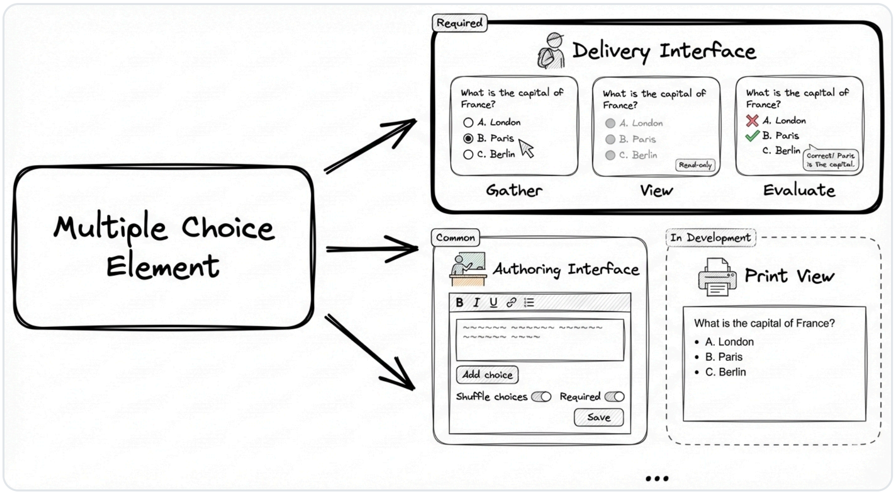
- **Gather mode**: Students can select answers and interact
- **View mode**: Read-only display without interaction
- **Evaluate mode**: Shows correctness, feedback, and correct answers

Furthermore, most PIE elements come with (and element authors are encouraged to support):

Authoring Interface: A configuration UI where educators define questions. For a multiple-choice element, this means adding choices, marking correct answers, setting layouts (vertical/horizontal), configuring feedback, and adjusting difficulty. The authoring interface uses form controls, rich text editors, and visual configuration panels.

We are also working on widely supporting:

- **Print View**: A UI optimized for printing
- **List View**: UIs that give usable previews that can be used in lists and tables



Every element provides delivery, authoring, and evaluation interfaces

Comprehensive Question Type Library

PIE includes over 20 production-ready question types out of the box, spanning simple to highly complex interactions:

Simple Response Types

- Multiple choice (single/multi-select, with configurable layouts and choice indicators)
- Text entry (short answer with validation patterns)
- Extended text entry (essay responses with rich text formatting)
- Inline choice (dropdown selections embedded within text)
- Math inline (mathematical expression input with proper rendering)

Complex Matching and Categorization

- Match (connect items between two lists)
- Match list (table-based matching interface)
- Categorize (drag items into labeled categories)
- Placement ordering (arrange items in correct sequence)

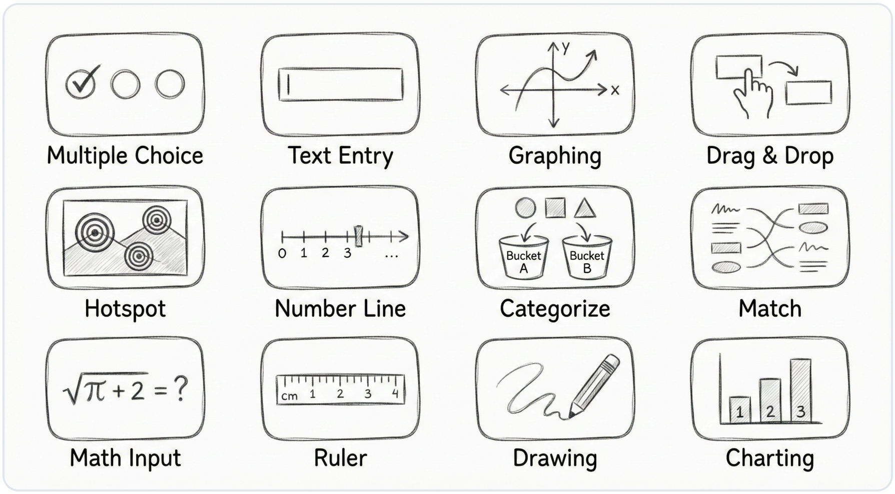
Visual and Graphical Responses

- Hotspot (click regions on images)
- Image cloze association (place labels on image locations)
- Drawing response (freehand drawing areas)
- Graphing (plot points, lines, curves, or functions on coordinate planes)
- Number line (identify points or ranges on number lines)

Assessment Support

- Passage (formatted reading passages with rich content)
- Rubric (scoring guides for constructed response evaluation)

Many elements include sophisticated features: partial credit scoring, choice-level feedback, randomization, accessibility support, and print-friendly rendering.



Examples of PIE question types in action

Building Custom Elements

Organizations with specialized assessment needs can build custom elements following PIE's established patterns. The framework provides:

- **Element boilerplate**: Templates and scaffolding for new question types
- **Shared component libraries**: Reusable UI components (prompts, feedback displays, math rendering)
- **Controller interface**: Framework-agnostic JavaScript/TypeScript interface for scoring and validation logic
- **Comprehensive examples**: 30+ reference implementations demonstrating patterns

Custom elements integrate seamlessly with PIE's existing infrastructure, automatically gaining features like accessibility support, print rendering, and player compatibility.

Accessibility Throughout

Every PIE element includes accessibility features as part of its core implementation:

- **Keyboard navigation**: All interactions accessible via keyboard (tab navigation, arrow keys, enter/space activation)

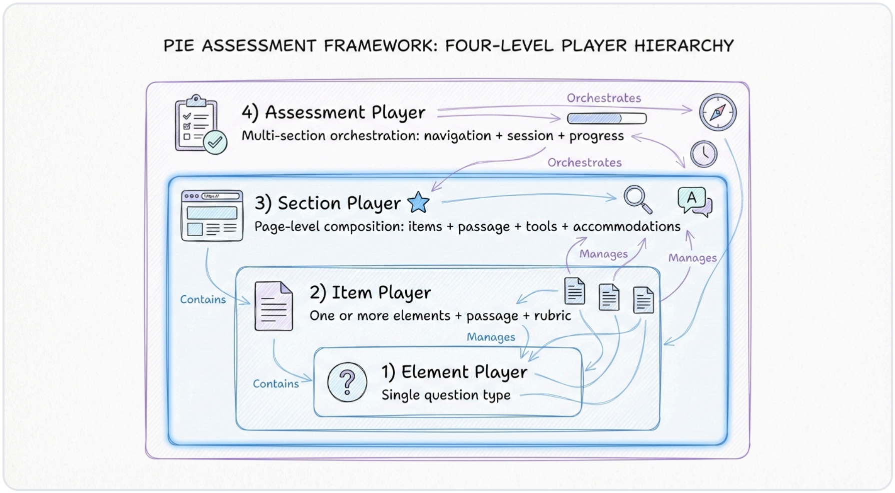
- **Screen reader support**: Semantic HTML with proper ARIA labels, roles, and live regions
- **Focus management**: Logical focus order and visible focus indicators
- **High contrast compatibility**: Elements render correctly with high-contrast themes
- **Magnification support**: Interfaces scale properly when users increase font sizes

These features aren't optional configurations—they're built into each element's design and tested as part of the development process.

SECTION 03

Players: From Elements to Complete Assessments

While elements represent individual question types, **players** orchestrate how elements are rendered, managed, and integrated into assessment experiences. PIE provides players at four levels, each independently usable depending on your architecture needs. You can adopt just the item player for embedding single questions, add the section player when you need page-level composition with tools, or use the full assessment player for multi-section test delivery.


PIE ASSESSMENT FRAMEWORK: FOUR-LEVEL PLAYER HIERARCHY
4) Assessment Player
Multi-section orchestration: navigation + session + progress
Orchestrates
Orchestrates
3) Section Player
Page-level composition: items + passage + tools + accommodations
Manages
Manages
Contains
2) Item Player
One or more elements + passage + rubric
Manages
Contains
1) Element Player
Single question type

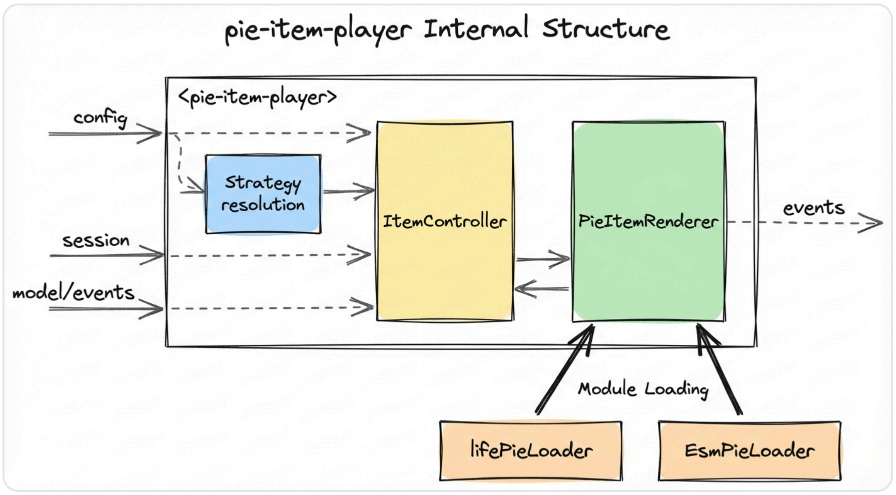
The four-level player hierarchy: Element, Item, Section, and Assessment

Four Levels of Players

PIE provides players at four levels of granularity, each implemented as an independent Web Component:

Element Player: Renders a single element (one question type). Primarily used for testing and development. This player loads the element's JavaScript bundle, manages its state, handles mode transitions, and emits events when responses change.

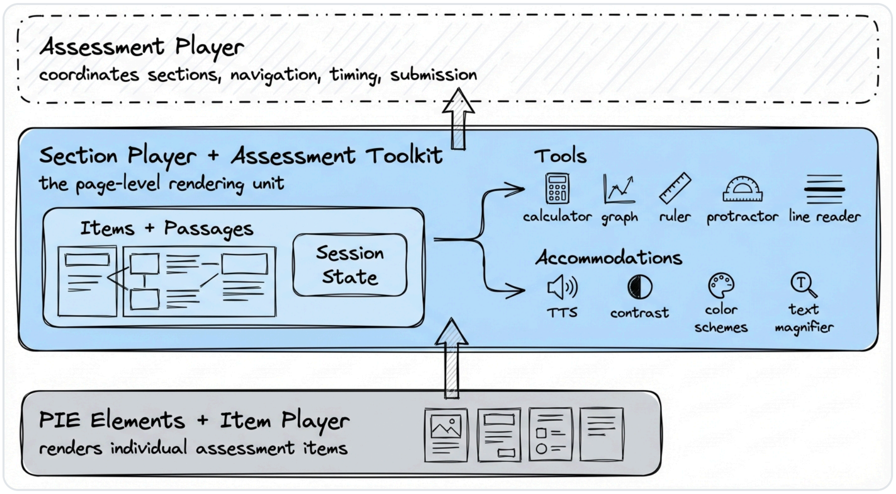
Item Player: Renders complete assessment items, which may contain one or more question elements, and optionally passage content and rubrics. The item player is a single unified component (`<pie-item-player>`) with a `strategy` attribute that controls how PIE element bundles are loaded. It manages state across elements, handles session deduplication, and supports both delivery and authoring modes in one package. Internally, the item player resolves the loading strategy, delegates to an `ItemController` for state management, and hands off rendering to a `PieItemRenderer` that instantiates the actual custom elements.



Inside the item player: strategy resolution, ItemController, PieItemRenderer, and pluggable module loaders

Section Player: The rendering workhorse of the PIE stack. A section maps to a page —it groups related content (a passage with its items, or a set of standalone items) into a single view with shared tools, session state, and navigation scope. The section player composes item players with toolbars, passages, and accommodation features into a coherent page-level experience. Two layout variants are available: `pie-section-player-splitpane` (passage on the left, items on the right) and `pie-section-player-vertical` (stacked layout). The section player integrates with the ToolkitCoordinator to coordinate tools like calculators, text-to-speech, and annotation at the section level.

Assessment Player: An orchestration layer above the section player. It coordinates which section is active, manages assessment-level session continuity across sections, provides navigation controls (back/next), and handles the delivery lifecycle. The assessment player does not render content itself—the section player remains the rendering engine. It routes between pages, tracks progress, and aggregates section sessions into an assessment session.


Assessment Player
coordinates sections, navigation, timing, submission
Section Player + Assessment Toolkit
the page-level rendering unit
Items + Passages
Session State
Tools
calculator
graph
ruler
protractor
line reader
Accommodations
TTS
contrast
color schemes
text magnifier
PIE Elements + Item Player
renders individual assessment items

From item rendering through section composition to assessment orchestration

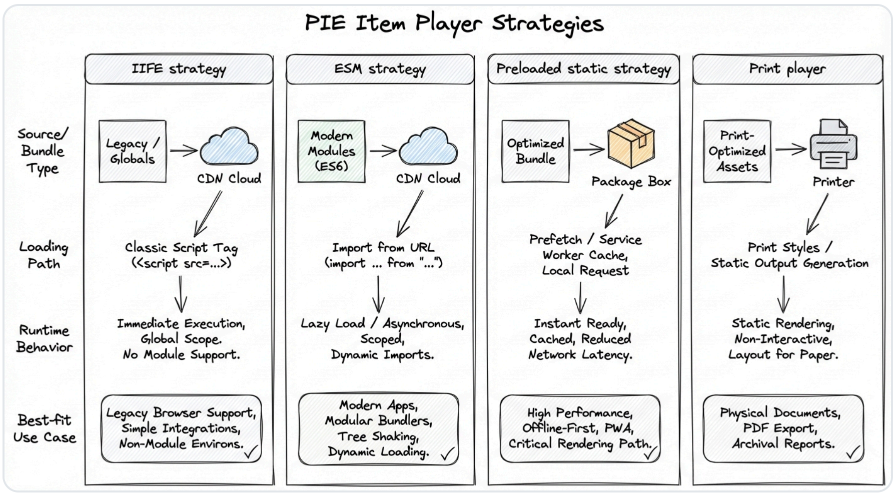
Item Player Loading Strategies

The item player (`<pie-item-player>`) uses a `strategy` attribute to control how PIE element bundles are fetched and registered. Three strategies are available, all producing identical rendering behavior:

IIFE Strategy (`strategy="iife"` , default): Loads pre-bundled element packages dynamically from a bundle hosting service by injecting `<script>` tags. This is the most widely deployed strategy and the safe default for any production system today. It supports older browsers and existing PIE deployments. Trade-off: large bundles with limited cross-element caching.

ESM Strategy (`strategy="esm"`): Uses native browser `import()` for dynamic module loading from an ESM CDN. Modules are fetched concurrently and cached by the browser's native module cache, shared dependencies are deduplicated at the module graph level, and the format supports tree-shaking and source maps. Offers roughly 85% bundle size reduction compared to IIFE. The PIE team is targeting ESM as the primary strategy by end of 2026.

Preloaded Strategy (`strategy="preloaded"`): All required PIE custom elements are pre-bundled at build time—no network requests at runtime. This trades flexibility for zero-latency rendering: useful for offline environments, strict performance budgets, or controlled test harnesses. Updating element versions requires a redeployment.

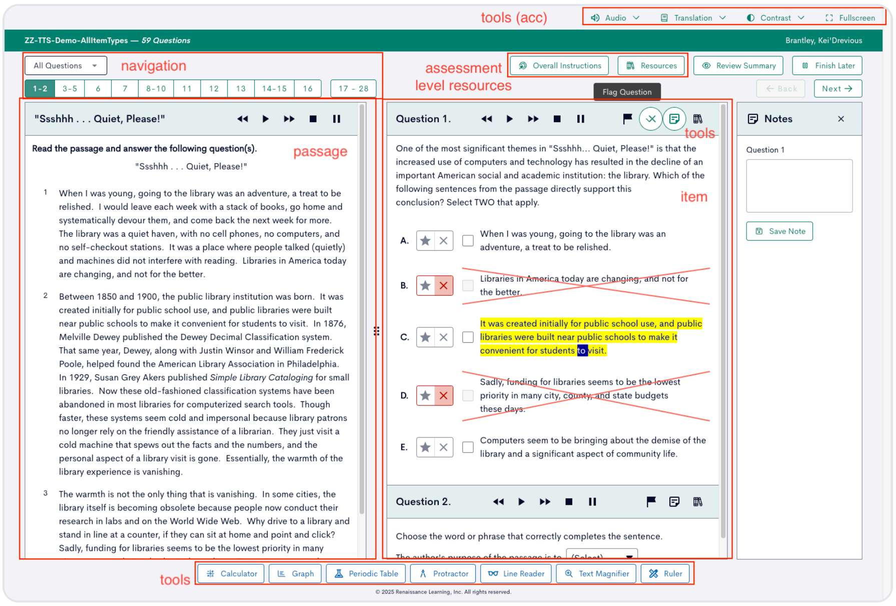


PIE Item Player Strategies
IIFE strategy
ESM strategy
Preloaded static strategy
Print player
Source/ Bundle Type
Loading Path
Runtime Behavior
Best-fit Use Case
Legacy / Globals
CDN Cloud
Classic Script Tag (<script src=...>)
Immediate Execution, Global Scope. No Module Support.
Legacy Browser Support, Simple Integrations, Non-Module Environs.
Modern Modules (ES6)
CDN Cloud
Import from URL (import ... from "...")
Lazy Load / Asynchronous, Scoped, Dynamic Imports.
Modern Apps, Modular Bundlers, Tree Shaking, Dynamic Loading.
Optimized Bundle
Package Box
Prefetch / Service Worker Cache, Local Request
Instant Ready, Cached, Reduced Network Latency.
High Performance, Offline-First, PWA, Critical Rendering Path.
Print-Optimized Assets
Printer
Print Styles / Static Output Generation
Static Rendering, Non-Interactive, Layout for Paper.
Physical Documents, PDF Export, Archival Reports.

Three loading strategies: IIFE, ESM, and Preloaded

Section Player: The Page-Level Composition Layer

Step back and look at what a real assessment screen looks like in practice:



"Ssshhh . . . Quiet, Please!"

Read the passage and answer the following question(s).

"Ssshhh . . . Quiet, Please!"

1 When I was young, going to the library was an adventure, a treat to be relished. I would leave each week with a stack of books, go home and systematically devour them, and come back the next week for more. The library was a quiet haven, with no cell phones, no computers, and no self-checkout stations. It was a place where people talked (quietly) and machines did not interfere with reading. Libraries in America today are changing, and not for the better.

2 Between 1850 and 1900, the public library institution was born. It was created initially for public school use, and public libraries were built near public schools to make it convenient for students to visit. In 1876, Melville Dewey published the Dewey Decimal Classification system. That same year, Dewey, along with Justin Winsor and William Frederick Poole, helped found the American Library Association in Philadelphia. In 1929, Susan Grey Akers published *Simple Library Cataloging* for small libraries. Now these old-fashioned classification systems have been abandoned in most libraries for computerized search tools. Though faster, these systems seem cold and impersonal because library patrons no longer rely on the friendly assistance of a librarian. They just visit a cold machine that spews out the facts and the numbers, and the personal aspect of a library visit is gone. Essentially, the warmth of the library experience is vanishing.

3 The warmth is not the only thing that is vanishing. In some cities, the library itself is becoming obsolete because people now conduct their research in labs and on the World Wide Web. Why drive to a library and stand in line at a counter, if they can sit at home and point and click? Sadly, funding for libraries seems to be the lowest priority in many

Question 1.

One of the most significant themes in "Ssshhh... Quiet, Please!" is that the increased use of computers and technology has resulted in the decline of an important American social and academic institution: the library. Which of the following sentences from the passage directly support this conclusion? Select TWO that apply.

A. When I was young, going to the library was an adventure, a treat to be relished.

B. ~~Libraries in America today are changing, and not for the better.~~

C. It was created initially for public school use, and public libraries were built near public schools to make it convenient for students to visit.

D. ~~Sadly, funding for libraries seems to be the lowest priority in many city, county, and state budgets these days.~~

E. Computers seem to be bringing about the demise of the library and a significant aspect of community life.

Question 2.

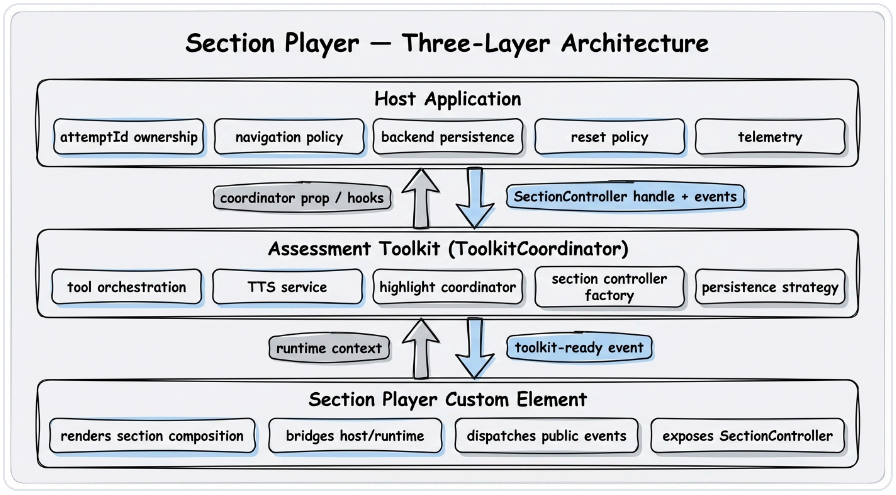
Choose the word or phrase that correctly completes the sentence.

A section as students see it — passage, items, tools, accommodations, and navigation composed into a single view

There's a reading passage on the left paired with questions on the right. A toolbar at the bottom offers section-wide tools: calculator, graph, periodic table, protractor, line reader, ruler. Each item has its own controls—text-to-speech playback, annotation, answer elimination. Accommodation controls manage audio, contrast, and fullscreen mode. All of this needs coordination, shared state, and a coherent lifecycle.

Every team that builds beyond the item player level ends up rebuilding some version of this composition. The section player exists to provide that layer as a ready-made, well-tested foundation—so integration teams can focus on their product's unique concerns rather than re-solving passage-item layout, tool coordination, session persistence, and accessibility plumbing.

The system is built around three distinct layers:



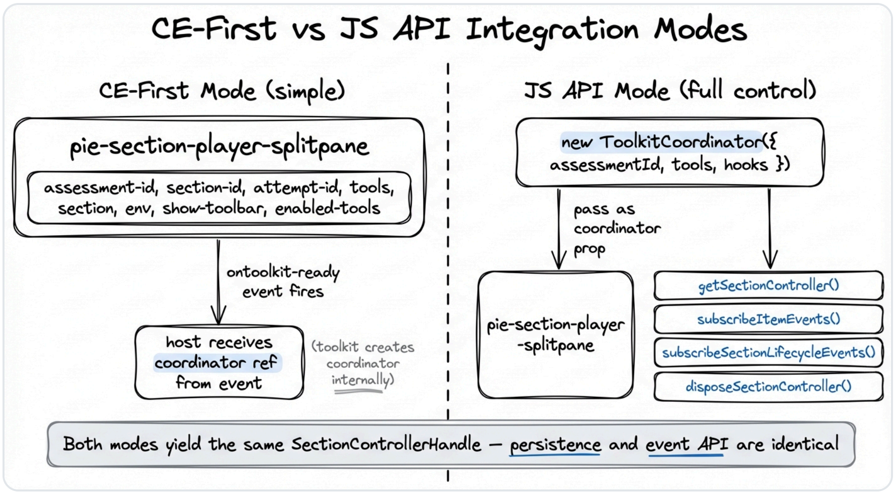
Three-layer architecture: Section Player CE, ToolkitCoordinator, and Host Application

Section Player custom element: Renders the section composition—items, passages, toolbars—and bridges between the host runtime contracts and the internal rendering engine. It dispatches lifecycle events and exposes a controller handle for programmatic session access.

Assessment Toolkit (ToolkitCoordinator): Orchestrates all toolkit services: tool availability, TTS, highlight coordination, accessibility catalogs, and section controller lifecycle. It is the single authoritative service hub for an assessment context.

Host Application: Owns everything with business meaning: attempt identity, navigation continuity across page loads, backend persistence, reset policy, and telemetry.

The section player supports two integration modes: **CE-first** (pass tool and section configuration as element attributes, the player creates the ToolkitCoordinator internally) and **JS API** (construct the ToolkitCoordinator yourself for full lifecycle control). Both produce identical runtime behavior and yield the same `SectionControllerHandle` — the persistence and event API is identical regardless of which mode you choose.

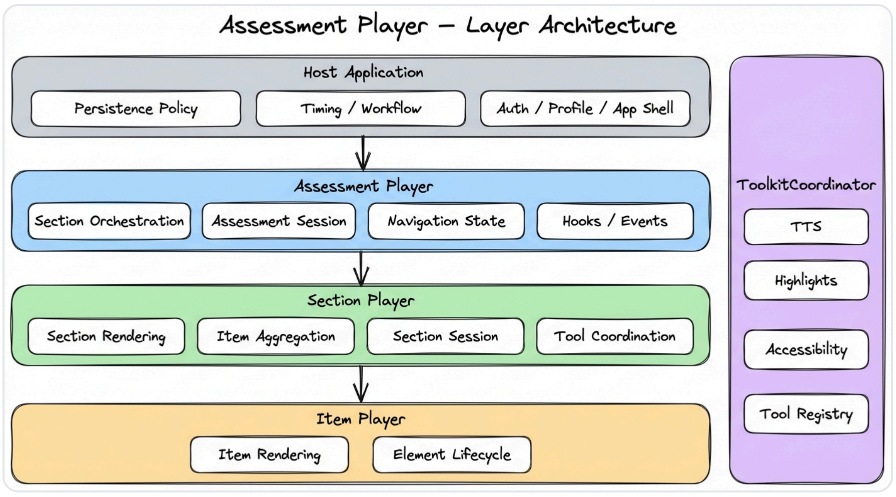

CE-First vs JS API Integration Modes
CE-First Mode (simple)
pie-section-player-splitpane
assessment-id, section-id, attempt-id, tools, section, env, show-toolbar, enabled-tools
ontoolkit-ready event fires
host receives coordinator ref from event
(toolkit creates coordinator internally)
JS API Mode (full control)
new ToolkitCoordinator({ assessmentId, tools, hooks })
pass as coordinator prop
pie-section-player -splitpane
getSectionController()
subscribeItemEvents()
subscribeSectionLifecycleEvents()
disposeSectionController()
Both modes yield the same SectionControllerHandle — persistence and event API are identical

Two integration modes, one API contract — both yield the same SectionControllerHandle

Assessment Player: Multi-Section Orchestration

Above the section layer, the assessment player coordinates sections—navigation between them, assessment-level session continuity, progress tracking, and submission workflow. It routes between pages; it doesn't render them. The section player remains the rendering workhorse.

The assessment player uses a QTI-inspired content model: an assessment definition containing sections (or nested test parts with sections). It flattens this structure into a linear delivery plan and navigates by index. Each section is passed to a section player when it becomes active. Session state is layered recursively: item sessions live inside section sessions, which live inside the assessment session aggregate.



Assessment Player — orchestration layer above the section player

Like the section player, the assessment player supports both CE-first and JS API integration modes, with cancelable navigation events that let the host implement custom gating policies (time limits, completion requirements, review restrictions) without the framework prescribing those rules.

Web Component Standard

All PIE players are implemented as Web Components—specifically, HTML5 custom elements. This means they integrate into any web page or application as standard HTML tags:

HTML

```html
<pie-item-player
  strategy="iife"
  config="..."
  env='{"mode":"gather","role":"student"}'
  session='{"id":"s1","data":[]}'>
</pie-item-player>
```

For section-level delivery with tools and accommodations:

HTML

```
<pie-section-player-splitpane
  assessment-id="my-assessment-001"
  section-id="section-a"
  attempt-id="attempt-xyz"
  show-toolbar="true"
  enabled-tools="theme,graph,periodicTable,lineReader"
  player-type="iife">
</pie-section-player-splitpane>
```

Under the hood, players are Web Components that communicate through standard DOM APIs—properties, events, and methods. For the element and item players, this DOM-level interface is the primary API. However, for most production integrations using the section player or assessment player, raw DOM events should be considered an implementation detail. Instead, the **section controller** provides a well-defined, typed events and methods interface as the primary integration surface:

- **Controller handle**: When the section player initializes, it exposes a `SectionControllerHandle` with typed methods for session access (`getSession()`), persistence (`persist()`), hydration (`hydrate()`), and lifecycle management
- **Controller events**: The section controller emits strongly typed events (`onSessionChanged`, `onReady`, `onError`) through callback subscriptions rather than raw DOM event listeners—giving host applications a stable, versioned contract
- **Properties**: Pass configuration (section content, tools, environment) via element attributes or JavaScript object properties
- **Assessment controller**: At the assessment level, a similar controller abstraction manages cross-section navigation, progress tracking, and assessment session aggregation

This layered approach means you use DOM APIs directly only if you need fine-grained element or item player control. For section and assessment delivery, the controller API is the contract—making integrations more robust and less coupled to internal DOM structure.

SECTION 04

Assessment Toolkit: Tools and Accommodations

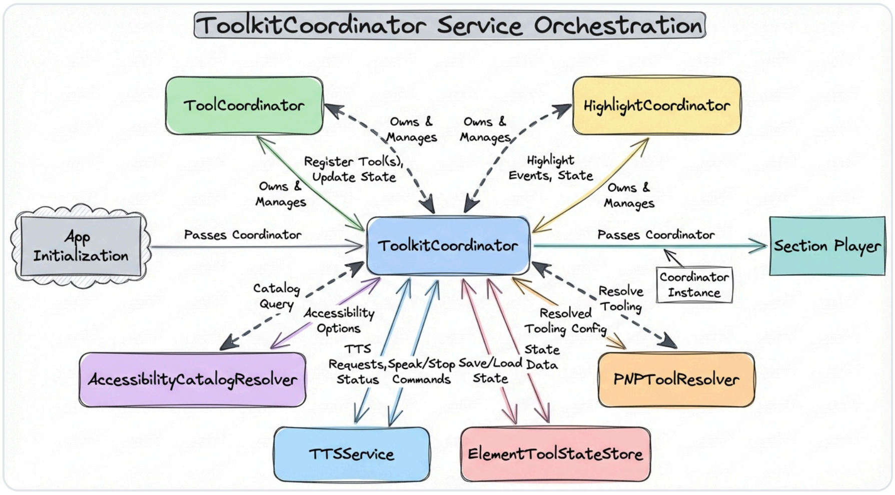
Real-world assessments require more than just question rendering. Students need calculators, text-to-speech, annotation tools, and accommodations mandated by IEPs (Individualized Education Programs) and 504 plans. The PIE Assessment Toolkit provides these capabilities as composable services coordinated through a centralized hub—the **ToolkitCoordinator**.

ToolkitCoordinator: Composable Architecture

The toolkit is designed as a toolkit, not a framework—products use only what they need. Rendering items without tools? Use just the player. Need calculator and ruler? Add those tools. Full accommodation support with TTS, high contrast, and PNP profiles? Integrate the complete toolkit.

The **ToolkitCoordinator** is the single entry point for initializing and managing all toolkit services. Products construct one coordinator per assessment context, configure tools and providers, and pass it to the section player. The coordinator owns all services as public properties, so host applications can also access them directly when needed (for example, wiring a standalone TTS control panel or annotation toolbar that the host renders outside the player).

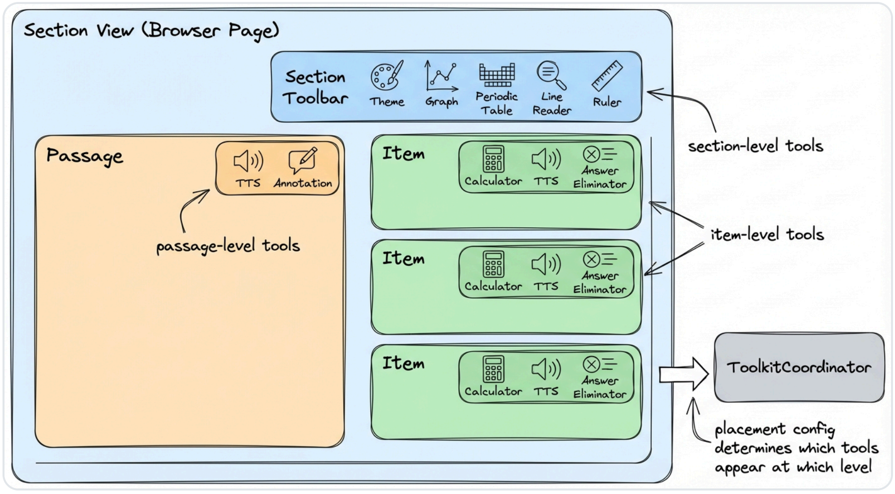
When connected to the section player, the toolkit automatically handles SSML extraction from passages and items, manages accessibility catalog lifecycle (add on load, clear on navigation), renders TTS tools inline in passage/item headers, coordinates z-index layering for tools, and synchronizes text highlighting with TTS playback.



ToolkitCoordinator orchestrates toolkit services and section player integration

Tool Placement

Tools are placed at three levels—section, item, and passage—each with its own toolbar scope. A calculator in the section toolbar is available across all items. Text-to-speech placed at the item level gets a playback button in each item header. Annotation tools at the passage level let students highlight within reading content. The ToolkitCoordinator drives which tools appear where based on a placement configuration.



Three placement levels: section toolbar, per-item toolbars, and per-passage toolbars

Tool Categories

Reference Tools

- **Calculator**: Scientific and graphing calculator modes (integrated with Desmos for graphing)
- **Ruler**: Metric and imperial measurements with dragging, rotation, and snap-to-grid
- **Protractor**: Angle measurement with center origin alignment
- **Graph**: Interactive coordinate plane
- **Periodic Table**: Chemistry reference

Accessibility Tools

- **Line Reader**: Masks content to highlight individual lines (reduces visual distraction)
- **Color Schemes**: High-contrast themes and accessibility overlays (black-on-white, white-on-black, custom palettes)
- **Theme Tool**: Student-facing light/dark mode toggle, separate from developer-controlled theme configuration
- **Text-to-Speech**: Reads questions aloud with word-level highlighting synchronized to audio playback, using pluggable providers (browser Web Speech API, AWS Polly, Google TTS)

Student Support Tools

- **Annotation Toolbar**: Highlight text in multiple colors, create underlines and notes
- **Answer Eliminator**: Cross out multiple-choice options to narrow choices

Each tool is implemented as a Web Component, enabling tools to be used independently or composed into toolbars and menus. In assessment terminology, "tools" (calculator, ruler) and "accommodations" (TTS, contrast modes) are often treated as separate categories—tools enabled per-assessment based on content, accommodations assigned per-student based on accessibility needs. The toolkit treats all of these uniformly at the framework level: same placement model, same provider configuration, same lifecycle. The policy distinction (which student sees which tools) is the host application's responsibility.

Service Architecture

The ToolkitCoordinator manages five core services that work together:

ToolCoordinator: Manages z-index layering and visibility for tools. When multiple tools are open (calculator, protractor, annotation toolbar), it ensures proper stacking order with reserved z-index ranges across five layers: content, non-modal tools, modal tools, control handles, and highlight infrastructure.

HighlightCoordinator: Manages simultaneous highlighting for different purposes using the CSS Custom Highlight API (a modern browser standard). Students can have persistent annotations in one color while TTS temporarily highlights the current word in another color. Zero DOM mutation—preserves framework virtual DOM, is screen reader friendly, and allows multiple highlights to overlap gracefully.

TTSService: Provides text-to-speech with synchronized word-level highlighting. The service abstracts TTS providers (browser Web Speech API, AWS Polly, Google TTS) behind a pluggable provider interface, handles playback state, and coordinates with HighlightCoordinator for real-time word tracking.

AccessibilityCatalogResolver: Resolves QTI 3.0-inspired accessibility catalogs with priority ordering: extracted SSML from content, item-level catalogs, and assessment-level catalogs. Supports pre-recorded audio, sign language videos, and braille alternatives.

ElementToolStateStore: Manages per-element ephemeral tool state (such as which annotation colors are active or calculator position) using globally unique composite keys. Tool state is kept separate from PIE session data—it's client-only and not sent to the server for scoring.

Additionally, a **PNPToolResolver** resolves tool availability using QTI 3.0 Personal Needs Profile (PNP) precedence rules, mapping student accommodation profiles to PIE tool IDs. The precedence hierarchy supports district blocks, test administration overrides, item restrictions, item requirements, district requirements, and PNP supports—ensuring that accommodation policy is applied correctly even when multiple configuration layers conflict.

The `<pie-theme>` **element** provides theming through a provider adapter model. It reads CSS variables from an existing design system (a DaisyUI adapter is built in) and maps them to PIE's `--pie-*` custom properties. Color schemes are accessibility overlays applied on top of the base theme, and students can select their preferred scheme at runtime via the color scheme toolbar tool.

WCAG 2.2 Compliance

The toolkit's accessibility features target WCAG 2.2 Level AA compliance:

- All tools are keyboard navigable with visible focus indicators
- Screen readers receive appropriate announcements via ARIA live regions
- Color contrast meets 4.5:1 minimum ratios
- Interactive elements meet minimum touch target sizes (44×44 pixels)
- Time limits include warnings and extension mechanisms
- Alternative text and captions support content accessibility

SECTION 05

Integration and Architecture

PIE's integration model separates frontend concerns (rendering, interaction, accessibility) from backend concerns (persistence, authentication, business logic). This section explains how these pieces connect and the architectural patterns that make PIE production-ready.

PIE's fundamental integration point is the Web Component standard. Players are distributed as custom HTML elements. For most production deployments, the **section player** is the primary integration surface—it composes item rendering with tools, accommodations, and session management in a single element. For multi-section assessments, the **assessment player** sits on top as the orchestrator. For simpler use cases (embedding a single question in a CMS), the item player can be used directly.

> INTEGRATION PRINCIPLE
>
> **Player and toolkit handle runtime mechanics; the host handles durable data and policy.** The section player and assessment player manage rendering, tool coordination, and session lifecycle. Your application owns attempt identity, backend persistence, navigation policy, timing, and business rules.

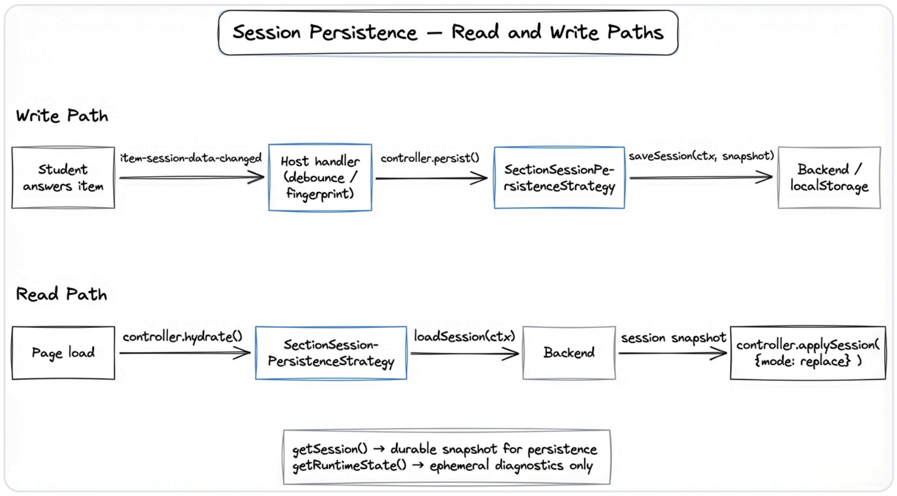
Your application communicates with PIE through the **section controller** (or assessment controller), which provides a well-defined, typed API on top of the underlying Web Component plumbing:

- **Setting properties**: Pass configuration (section content, tools, environment) via element attributes or JavaScript object properties
- **Reading state**: Access session snapshots via the controller handle (`getSession()` for persistence-safe data)

- **Subscribing to events**: The controller exposes typed event callbacks (`onSessionChanged`, `onReady`, `onError`, `onNavigationRequested`) rather than requiring raw DOM event listeners—giving your application a stable contract that won't break when internal component structure changes
- **Controller methods**: Programmatic access to `hydrate()`, `persist()`, `navigateNext()`, and more

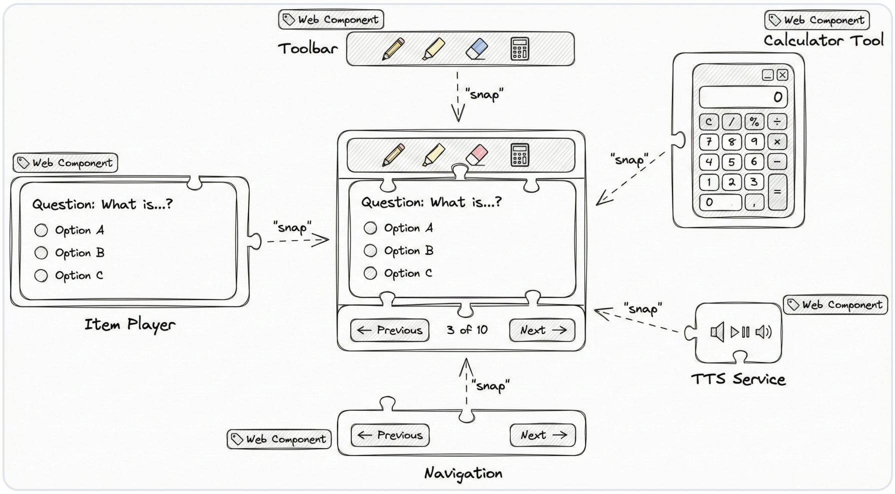
While DOM events flow under the hood (this is how Web Components work), the controller interface is the intended integration surface for section and assessment player clients. This matters because it decouples your integration from internal DOM structure—your code subscribes to `controller.onSessionChanged(...)` rather than listening for a `session-changed` event on a specific element.

A concrete example of this controller-mediated flow is session persistence. When a student answers a question, the host's `onSessionChanged` handler calls `controller.persist()`, which delegates to a `SectionSessionPersistenceStrategy` that the host provided at initialization. On page load, `controller.hydrate()` reverses the flow—loading a session snapshot from the backend and applying it to the player. The controller distinguishes between `getSession()` (a durable snapshot safe for persistence) and `getRuntimeState()` (ephemeral diagnostics only).


Session Persistence — Read and Write Paths
Write Path
Student answers item
item-session-data-changed
Host handler (debounce / fingerprint)
controller.persist()
SectionSessionPersistenceStrategy
saveSession(ctx, snapshot)
Backend / localStorage
Read Path
Page load
controller.hydrate()
SectionSessionPersistenceStrategy
loadSession(ctx)
Backend
session snapshot
controller.applySession({mode: replace})
getSession() → durable snapshot for persistence
getRuntimeState() → ephemeral diagnostics only

Session persistence — read and write paths through the SectionSessionPersistenceStrategy

Because Web Components are browser-native, they work consistently across all modern frameworks. A React application, a Vue application, and a vanilla JavaScript application integrate with PIE identically.



Web Components assembly — PIE integrates with any framework

Separation of Concerns

PIE and your backend divide responsibilities clearly:

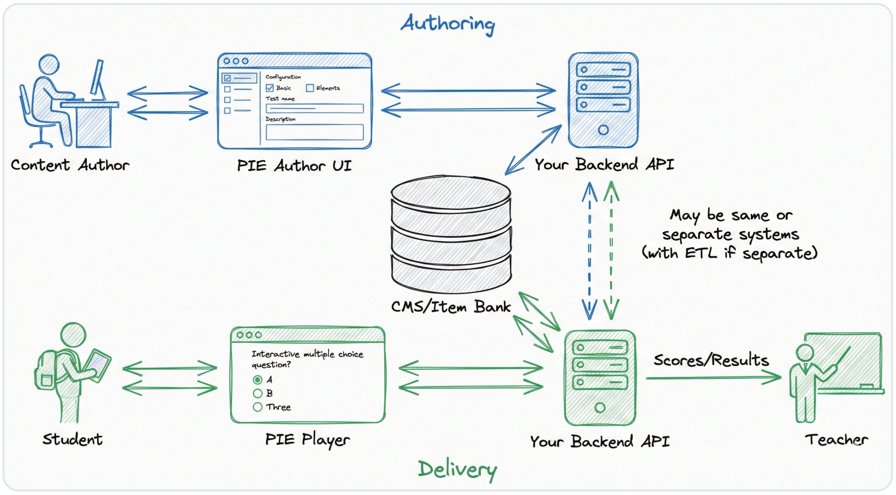
YOUR BACKEND RESPONSIBILITIES

- **Item Storage**: Maintain your item bank or CMS storing item configurations (JSON documents defining questions). This may be a unified content repository or separate authoring and production systems.
- **Authentication & Authorization**: Validate users (content authors, administrators, teachers, students), enforce permissions, manage sessions
- **Assessment Administration**: Track which students take which tests, enforce time limits, record attempt counts, manage test assignments
- **Session Persistence**: Store student responses as they work through items. PIE's players provide persistence hooks—you supply the strategy (your backend API), and the player handles the lifecycle (when to load, save, and clear).
- **Scoring**: Execute PIE controller logic server-side for high-stakes assessments, persist scores
- **Reporting**: Generate score reports, analytics, and learning insights for teachers and administrators

PIE RESPONSIBILITIES

- **Rendering**: Display items in browsers with correct layout, styling, and behavior
- **Composition**: Compose items, passages, and tools into page-level views (section player) and coordinate multi-section navigation (assessment player)
- **Interaction**: Capture student responses through accessible, intuitive interfaces
- **Authoring**: Provide configuration UIs for educators to create and modify content
- **Accessibility**: Ensure WCAG compliance through proper markup, keyboard support, and screen reader compatibility
- **Tools & Accommodations**: Coordinate calculators, TTS, annotation, color schemes, and accommodation features through the ToolkitCoordinator
- **Session Lifecycle**: Manage session hydration, persistence triggers, and replay within the player's runtime scope
- **Client-side Scoring** (optional): Calculate scores in browser for low-stakes or immediate feedback scenarios

This division means PIE integrates with virtually any backend technology. Your authoring CMS and production delivery system might be the same infrastructure or entirely separate (with ETL processes synchronizing content between them). Either way, PIE works because it speaks JSON and emits standard events—your backend architecture remains your choice.



Integration flow between PIE components and your backend

MVVM Security Model

PIE implements a Model-View-ViewModel pattern with controllers acting as security gateways. This pattern is crucial for high-stakes assessments where answer key security matters.

Controllers are JavaScript/TypeScript functions that transform item models based on context:

- `model(question, session, env)`: Returns view model for rendering
- `outcome(question, session, env)`: Returns score and feedback

The `env` parameter specifies context:

- **role**: "student" or "instructor"
- **mode**: "gather" (answering), "view" (review), "evaluate" (show correctness)

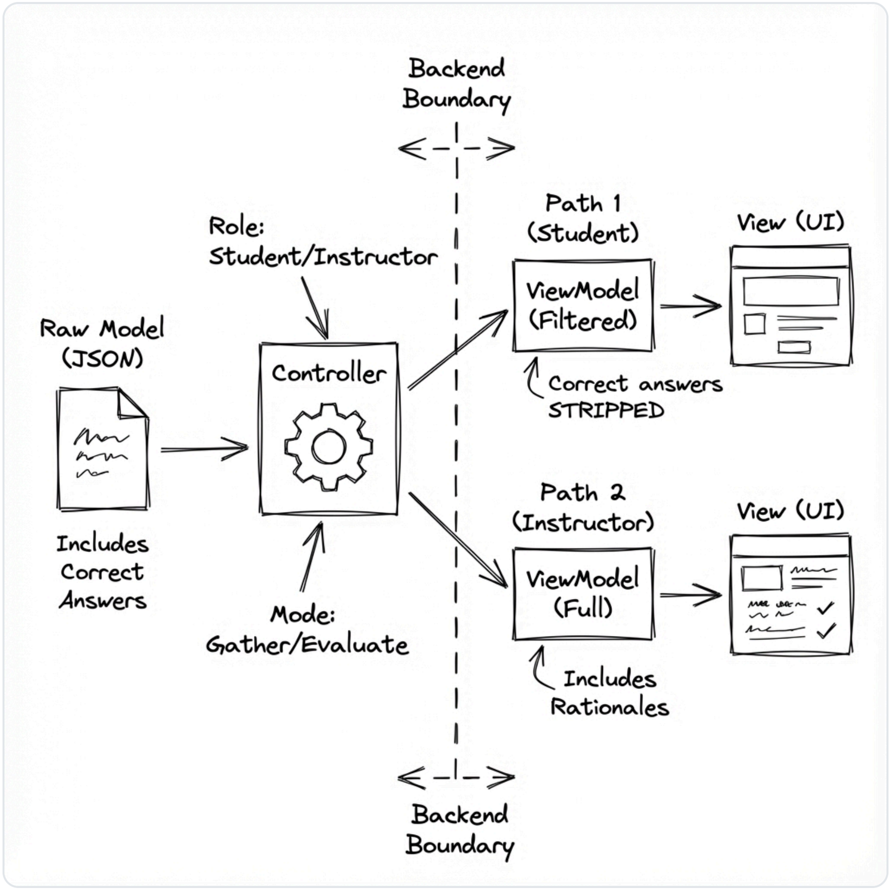
Controllers apply different transformations based on role and mode:

- Students in "gather" mode receive models without correct answers
- Instructors in "evaluate" mode receive models with correct answers and rationale
- Students in "evaluate" mode receive models with feedback but possibly limited answer details

Client-side execution (development, low-stakes): Controllers run in the browser. This enables rapid development, immediate feedback, and works well for practice quizzes or formative assessments where answer security isn't critical.

Server-side execution (production, high-stakes): Controllers run on your backend. Your server receives the student's session data, executes the controller, calculates the score, and returns only the outcome (score, feedback). Correct answers never reach the browser, making tampering ineffective.

This flexibility lets organizations choose appropriate security levels. A corporate training quiz might run controllers client-side for simplicity. A state accountability assessment runs controllers server-side for security.

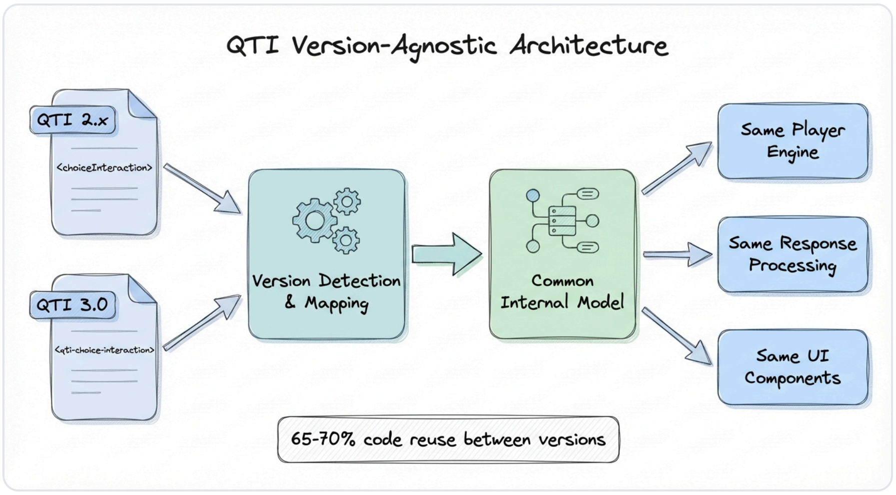


MVVM pattern — controllers as security gateways for client-side and server-side execution

Standards and Interoperability

PIE integrates with established assessment standards through dedicated tooling and QTI-inspired architectural patterns:

QTI (Question and Test Interoperability): PIE-QTI, a sibling project, provides production-ready QTI 2.x item and assessment players supporting all 21 standard interaction types and 45+ response processing operators. A version-agnostic architecture auto-detects QTI version from XML (namespace, root element, version attribute) and maps both QTI 2.x (camelCase) and QTI 3.0 (kebab-case with `qti-` prefix) content to a common internal model, achieving 65–70% code reuse between versions. Bidirectional PIE-to-QTI transforms are under active development, with a plugin-based transform engine, CLI, and web application. QTI 3.0 concepts also influenced PIE's player architecture directly: the assessment toolkit uses QTI 3.0 Personal Needs Profiles (PNP) for accommodation resolution, accessibility catalogs for alternative content, and context declarations for cross-item shared state. The section and assessment models in PIE's players are QTI-inspired, making content naturally mappable between the two ecosystems.



QTI Version-Agnostic Architecture — both QTI 2.x and 3.0 map to a common internal model

LTI (Learning Tools Interoperability): LTI support is planned as an upcoming addition to pie-players, enabling PIE-based assessments to be embedded as LTI tools within LMS platforms like Canvas, Moodle, and Blackboard. No official LTI integration exists yet, but the player architecture is designed with this integration in mind.

SECTION 06

Production-Proven at Scale

Renaissance Learning, a global educational technology company serving approximately 40% of U.S. schools, has standardized on PIE as their assessment item rendering framework. This deployment processes over hundreds of millions of student interactions annually.

Renaissance operates PIE across multiple products with different technical stacks, demonstrating PIE's integration flexibility. Organizations evaluating PIE can reference real-world evidence of its capabilities rather than relying on vendor claims or theoretical projections.

The framework's eight-year development history reflects continuous refinement based on production feedback. Features like the section player composition model, the ToolkitCoordinator, the multi-strategy item player architecture, and QTI-inspired accommodation resolution all emerged from real operational requirements, not speculative design.

For organizations building item banks or assessment platforms, this production heritage matters. Choosing PIE means building on a foundation that has proven capable of educational scale, not hoping an emerging framework will mature in time.

Final Thought

Whether you are thinking of building or improving your CMS or item bank solution, or whether you are just hoping to build tests directly, you should consider PIE as a supporting technology.

The only thing we haven't really done so far is widely advertise ourselves. Hopefully a primer like this and many of the new exciting features planned to be delivered in 2026 will help more educational companies find their way to our framework, use it to their and their students' benefit, and work together to make PIE even better.

PIE — Portable Interactions and Elements

Open Source Assessment Framework · Sponsored by Renaissance Learning