# ADR-009: Server-Side Rendering (Jinja2)

**Submitters**

- Luke Doyle (D00255656)
- Hafsa Moin (D00256764)

### Change Log

- approved 2026-02-13

### Referenced Use Case(s)

- UVote Frontend Rendering Requirements - Accessible, low-friction voter interface with WCAG 2.1 Level AA compliance for the UVote election system.

### Context

The UVote frontend must achieve WCAG 2.1 Level AA accessibility compliance as a mandatory project requirement. The rendering approach fundamentally affects the baseline accessibility of the application, the development effort required to achieve compliance, and the voter experience, particularly first paint time and JavaScript dependency.

The project proposal identifies accessibility as one of three critical barriers to adoption for online voting systems. Target users include voters with visual, auditory, motor, and cognitive impairments, as well as elderly voters with low digital literacy. The voting interface is fundamentally form-based: voters receive an email link, see a list of candidates, select one, and submit.

Jinja2 was used in a Year 2 DkIT project for the DevOps module. That prior experience gave both submitters working familiarity with the templating syntax, server-side rendering patterns, and Jinja2's integration with Python web frameworks, making it a natural and low-risk choice for the UVote frontend.

The team had no interest in adopting a JavaScript frontend framework such as React or Vue. The reasons were threefold. First, neither team member has a background in JavaScript or TypeScript, and learning a new language on top of the existing project scope was not feasible within the timeline. Second, the npm ecosystem carries documented security risks: supply chain attacks and malicious packages in the npm registry are a recurring concern that the team did not want to introduce into the project. Third, React's client-side rendering model requires significant additional work to meet the accessibility and security requirements of this project on top of a language learning investment.

The accidental monolith referenced in ADR-008 included React as part of its frontend. This was not a deliberate team choice, and the DevOps module lecturer identified it as a poor fit for the project. The move to microservices was used as the point to remove React entirely and adopt server-side rendering with Jinja2 throughout.

### Proposed Design

**Services and modules impacted:**

The frontend-service and voting-service each maintain their own Jinja2 template directories. Templates are rendered by FastAPI route handlers via Starlette's `Jinja2Templates` integration. Static files are served via Starlette's `StaticFiles` middleware.

Template structure:

```
templates/
├── base.html             # Base layout (nav, footer, accessibility features)
├── verify_identity.html  # DOB verification form
├── vote.html             # Ballot with candidate options
├── vote_success.html     # Confirmation with receipt token
```

```
├── vote_error.html       # Error messages
└── vote_verified.html    # Receipt verification page

```

Accessibility features implemented in the base template:

- `lang="en"` on the `<html>` element
- Skip navigation link for keyboard users
- Focus indicators on all interactive elements
- Semantic elements (`<main>`, `<nav>`, `<form>`, `<label>`, `<fieldset>`)
- ARIA live regions for dynamic status messages

**Configuration:**

- Template engine: Jinja2 via Starlette's `Jinja2Templates`
- Static files: served via Starlette's `StaticFiles` middleware
- CSS: custom stylesheet with responsive breakpoints
- No JavaScript build process: templates use plain HTML and CSS

**Integration points:**

- ADR-001 (FastAPI): templates rendered by FastAPI route handlers
- ADR-008 (Microservices): frontend-service and voting-service each maintain their own template directories
- ADR-005 (Token-Based Auth): voting flow begins with a token in the URL, rendered into the initial template form

### Considerations

**React SPA (not selected):** React provides rich interactivity and a large ecosystem. It was not selected for several reasons. Neither team member has a background in JavaScript or TypeScript, and the learning investment was not compatible with the project timeline. The npm supply chain has a documented and ongoing history of malicious packages and security incidents, which the team did not want to introduce as a dependency. React's client-side rendering model also requires extensive additional work to achieve WCAG AA compliance: screen reader announcements are delayed until JavaScript executes, focus management must be implemented manually for every page transition, and extensive ARIA attributes are needed to replicate what semantic HTML provides by default. React was present in the accidental monolith but was identified by the DevOps module lecturer as a poor fit for the project and was removed when the architecture moved to microservices.

**Vue.js SPA (not selected):** Vue.js has a simpler API than React and better accessibility documentation. It was not selected because it carries the same structural limitations for this use case: JavaScript dependency, delayed first paint, and manual focus management. The npm security concerns apply equally.

**Static site with API calls (not selected):** Pre-built static HTML with JavaScript `fetch()` calls for dynamic content requires JavaScript for any meaningful interactivity, inheriting the accessibility concerns of a full SPA without the interactivity benefits. It also requires a separate deployment from backend services and CORS configuration between the static file server and the backend.

**Limited interactivity:** Jinja2 templates produce full page reloads on navigation. For a voting system this is acceptable. The voter-facing flow is a three-step linear sequence (verify identity, see ballot, submit vote) where page reloads are natural transitions rather than interruptions.

**No client-side validation:** Form validation occurs on the server, requiring a full round-trip for error messages. Progressive enhancement JavaScript can be added in Stage 2 for non-essential client-side validation without breaking the base server-rendered experience.

**Perceived as less modern:** Server-side rendering is sometimes characterised as an older approach relative to SPAs. For this project, accessibility compliance and supply chain security are higher priorities than perceived technical modernity.

### Decision

Server-side rendering with Jinja2 was selected as the frontend approach for UVote.

The decision was shaped by three factors. The first is prior familiarity: Jinja2 was used in a Year 2 DkIT project for the DevOps module, giving the team direct experience with its syntax and integration patterns. This removed the learning curve that any JavaScript framework would have introduced. The DevOps module lecturer also identified React as a poor fit for the project when it appeared in the early monolith, reinforcing the decision to move away from it.

The second factor is security. The npm ecosystem's history of supply chain attacks and malicious packages was a genuine concern for the team. Jinja2 is a pure Python library with no npm dependency chain, which keeps the project's dependency surface simple and within the team's existing tooling.

The third factor is accessibility. Server-rendered HTML with semantic elements ( `<form>` , `<label>` , `<button>` , `<fieldset>` ) is inherently accessible. Screen readers announce content immediately on load without waiting for JavaScript to execute. Browser-native focus management works correctly by default on page navigation. Standard HTML form POST handles vote submission without JavaScript event handlers. Achieving equivalent accessibility with a React or Vue SPA would require significant additional engineering on top of the language learning investment, and would still carry the npm security risk.

The trade-off of reduced interactivity relative to a SPA is accepted. A voting system is a form-submission application. It does not require real-time updates, client-side state management, or complex transitions. Page reloads between the three steps of the voting flow are natural and appropriate.

Progressive enhancement JavaScript may be added in Stage 2 for non-essential features such as client-side form validation and loading indicators, in a way that enhances but does not replace the server-rendered base experience.

A formal WCAG AA audit using automated tools and manual screen reader testing is planned for Stage 2.

### Other Related ADRs

- ADR-001: Python FastAPI Backend - Jinja2 support provided via Starlette integration
- ADR-008: Microservices Architecture - frontend-service and voting-service structure
- ADR-005: Token-Based Voter Authentication - token URL rendered into initial template form

### References

- Jinja2 Documentation
- WCAG 2.1 Quick Reference
- Starlette Jinja2 Templates
- EdgeX Foundry ADR Template