

PolPaL (Political Party Links)

Technical Stack and Clever Cloud as PaaS v1.1

Paul Bederke

As of October 2026

Table of Contents

1. Requirements 3

1.1 Technology Stack Overview 3

1.1.1 Node.js 3

1.1.2 Next.js 3

1.1.3 React 4

1.1.4 TypeScript 4

1.1.5 Tailwind CSS, shadcn/ui, and Radix UI 5

1.1.6 Zod 5

1.1.7 Prisma ORM 5

1.1.8 PostgreSQL 6

1.1.9 Authentication and Security Libraries 6

1.1.10 Logging 7

1.2 Compliance & Data Residency 7

1.2.1 Key GDPR Compliance Principles 7

1.2.2 Data Residency within the European Union 8

1.2.3 Data Processing Agreements and Documentation 8

2. Platform-as-a-Service (PaaS) provider 8

2.1 CleverCloud 9

2.1.1 Key Advantages 9

2.1.2 Use in Development and Production 10

2.2 CleverCloud Pricing Model 10

2.3 Expected Costs 11

3. Version history 11

Note: Most of this text was generated with Claude Sonnet 4.6 with input prompts describing the project and the software code of PolPaL. The generated text was proof-read and adjusted if deemed necessary.

1. Requirements

PolPaL (Political Party Links) is intended to operate as a modern web application, accessible via standard web browsers and designed for interactive user engagement. The technology stack includes Node.js, Next.js 16 (App Router), React 19, TypeScript 5, Tailwind CSS 4, shadcn/ui, Zod, Prisma ORM, and PostgreSQL.

To ensure compliance with the General Data Protection Regulation (GDPR), all data is securely stored within the European Union (EU), and appropriate technical and organizational measures are implemented to protect user privacy.

1.1 Technology Stack Overview

The PolPaL web application is built as a unified full-stack application using a modern JavaScript-based framework. All server-side logic — including API endpoints, authentication, and database access — is handled through Next.js Route Handlers; there is no separate backend server. The following components form the core of the stack.

1.1.1 Node.js

Node.js is a **JavaScript runtime environment** that allows JavaScript to be executed on the server side. It provides a non-blocking, event-driven architecture ideal for building scalable web applications. Node.js will serve as the foundation for server-side logic, handling API requests, routing, and application-level functionality.

Key roles in the project:

- Execution environment for the backend
- Handling requests to the database and APIs
- Supporting server-side rendering via Next.js

1.1.2 Next.js

Next.js (v16, App Router) is a React-based full-stack framework that serves as the single unified application layer for PolPaL. It provides both the React UI pages and server-side Route Handlers that act as the API layer.

Key advantages:

- App Router with file-based routing for both pages and API endpoints
- Server-side rendering (SSR) and React Server Components for fast, SEO-friendly pages
- Built-in middleware support (src/proxy.ts) for auth enforcement and CSP nonce injection
- Integrated TypeScript, image optimization, and Turbopack-powered dev server

Role in PolPaL:

- Hosts all frontend pages (src/app/) and backend Route Handlers (src/app/api/)
- Enforces authentication and Content Security Policy at the edge via the proxy
- Handles server-side data fetching and rendering

1.1.3 React

React (v19) is a **JavaScript library for building user interfaces**. It uses a component-based architecture that allows developers to build reusable UI elements with state and props management. PolPaL uses React 19 together with React Server Components (via the Next.js App Router). That moves the data-fetching logic to the server and reduces client-side JavaScript.

Usage in the application:

- Build interactive, modular frontend components
- Enable real-time UI updates and user interaction
- Power the frontend of the Next.js App Router

1.1.4 TypeScript

TypeScript (v6, strict mode) is a **strongly-typed superset of JavaScript** that compiles to plain JavaScript. It introduces type safety, interfaces, and modern tooling, which significantly improve code maintainability, refactoring capabilities, and developer productivity.

Benefits for the project:

- Catch bugs early through static type checking

- Enforce a consistent and scalable codebase across frontend and backend
- Improve developer collaboration and long-term maintainability
- Type-safe integration with Prisma ORM and Zod validation schemas

1.1.5 Tailwind CSS, shadcn/ui, and Radix UI

Tailwind CSS (v4) is a utility-first CSS framework that allows building custom user interfaces directly in markup without writing traditional CSS files. Combined with **shadcn/ui** — a collection of pre-built, accessible React components built on top of **Radix UI** primitives — it provides the full styling and component layer for PolPaL.

Key advantages:

- Consistent, responsive design with minimal custom CSS
- Accessible components (keyboard navigation, ARIA attributes) out of the box via Radix UI
- Lucide React provides a consistent icon set throughout the UI
- Recharts provides interactive, composable chart components for data visualization

Role in PolPaL:

- Style all pages and UI components uniformly
- Provide accessible form controls, dropdowns, dialogs, and data tables
- Render charts and statistics on data overview pages

1.1.6 Zod

Zod (v4) is a TypeScript-first **schema validation library** used to declare and enforce the shape of data at runtime. In PolPaL, Zod schemas are defined alongside each domain module and are used to validate all incoming API request payloads before any service or database call is made.

Role in PolPaL:

- Validate and parse request bodies in every Route Handler
- Provide a single source of truth for data shapes shared between frontend and backend
- Return structured, user-friendly validation errors without manual error handling code

1.1.7 Prisma ORM

Prisma (v7, with @prisma/adapter-pg) is an **Object-Relational Mapping (ORM) tool** for TypeScript and Node.js. It provides a type-safe and auto-generated database client, making database access intuitive and secure. The schema is split into one .prisma file per model under prisma/models/ and combined by a root schema.prisma.

Role in the project:

- Define and manage the PostgreSQL schema via the Prisma Schema Language
- Handle database queries with auto-completion and type safety
- Generate migrations and seed data
- Uses the pg driver adapter (connection pool, max 45 connections) for efficient connection handling

1.1.8 PostgreSQL

PostgreSQL (v17) is an open-source **relational database management system (RDBMS)** known for its stability, extensibility, and advanced features such as full-text search, JSON support, and robust ACID compliance. In production, the database runs as a managed service on Clever Cloud; in local development it runs in a Docker container defined in compose.yaml.

Use in PolPaL:

- Store structured data related to political parties, datasets, users, and audit logs
- Ensure data integrity and relationships across entities via foreign key constraints and Prisma migrations
- Support complex queries for data visualization and filtering
- Host the rate-limiter state table (used by rate-limiter-flexible) alongside application data

1.1.9 Authentication and Security Libraries

PolPaL implements authentication without a third-party auth provider, relying on a set of focused libraries:

- **jose** (v6) — signs and verifies **JSON Web Tokens** (HS256) stored in HTTP-only cookies (polpal_token, 6-hour lifetime)
- **bcrypt** (v6) — hashes user passwords before storage

- **rate-limiter-flexible** (v11) — enforces per-IP rate limits on sensitive endpoints (login, registration, password reset) with a PostgreSQL-backed counter store
- **nodemailer** (v10) — sends transactional emails (email verification, password reset, admin notifications)
- **disposable-email-detector** — rejects registrations from known disposable email domains

1.1.10 Logging

Pino (v10) is a high-performance **structured JSON logger** used for both application logging and security event tracking. A singleton logger instance is provided via src/lib/server/logger.server.ts. Security-relevant events (failed logins, rate-limit hits, suspicious requests) are recorded through a dedicated logSecurityEvent() helper with sliding-window alerting.

1.2 Compliance & Data Residency

The PolPaL web application strictly adheres to the **General Data Protection Regulation (GDPR)** to ensure the lawful, transparent, and secure processing of personal and potentially sensitive data. Given that the project may involve user interactions, contributor submissions, or tracking of usage patterns, compliance with GDPR is not only a legal requirement but also essential to building trust with users, collaborators, and stakeholders. The project will maintain internal documentation describing data flows, technical safeguards, and procedures for fulfilling data subject rights (e.g., data access, correction, or deletion requests).

1.2.1 Key GDPR Compliance Principles

The system design and deployment will follow the core principles of GDPR:

- **Data Minimization:** Only collect and store data that is strictly necessary for the functionality of the application.
- **Purpose Limitation:** Ensure data is collected for specified, legitimate purposes and not further processed in a manner incompatible with those purposes.
- **Integrity and Confidentiality:** Implement technical and organizational measures to protect personal data from unauthorized access, alteration, or loss.

- **Transparency and Accountability:** Clearly inform users about data collection practices through a privacy policy and, where necessary, obtain explicit consent.

1.2.2 Data Residency within the European Union

To comply with GDPR's requirements regarding cross-border data flows, **all user data is stored and processed exclusively within the European Union**. This includes the application's database, backups, and any associated content delivery services.

Implementation Measures:

- **EU-Based Hosting Providers:** The application is hosted on a platform that guarantees data storage in **EU-located data centers**.
- **No Transfer Outside EU:** No personal or usage data is transferred to or processed by services outside of the EU unless the receiving party provides an adequate level of data protection, in line with Article 45 of the GDPR (e.g., via Standard Contractual Clauses or adequacy decisions).
- **Encryption:** All data transmissions are secured via **HTTPS/TLS**, and sensitive data at rest will be encrypted when supported by the database or hosting provider.
- **Data Access Controls:** Role-based access control (RBAC) and audit trails are implemented to ensure that only authorized personnel can access sensitive data.
- **Backups and Disaster Recovery:** Regular encrypted backups are performed and stored in EU locations, with clear procedures for recovery and data retention timelines.

1.2.3 Data Processing Agreements and Documentation

User Rights and Consent Management

- Users will be informed about their rights under GDPR, including:
    - Right of access
    - Right to rectification
    - Right to erasure ("right to be forgotten")
    - Right to data portability
    - Right to restrict or object to processing
- Where applicable, a **consent management mechanism** (e.g., cookie banners or opt-in checkboxes) is implemented for non-essential data collection.

2. Platform-as-a-Service (PaaS) provider

Platform-as-a-Service (PaaS) is a category of cloud computing that provides developers with a ready-to-use environment for building, deploying, and managing applications without the complexity of managing the underlying infrastructure. PaaS platforms abstract away server provisioning, operating system management, and scaling, allowing developers to focus solely on writing and deploying application code.

Typical PaaS features include:

- Built-in support for popular programming languages and frameworks
- Integrated development tools and deployment pipelines
- Managed databases and services (e.g., PostgreSQL, Redis)
- Automatic scaling, backups, and monitoring
- Secure handling of environment variables and secrets

By using a PaaS, development teams can significantly reduce operational overhead, accelerate deployment cycles, and ensure greater consistency between development, staging, and production environments.

2.1 CleverCloud

To support the deployment and long-term operation of the PolPaL web application, the project utilizes CleverCloud as its primary hosting and deployment platform. CleverCloud is a European Platform-as-a-Service (PaaS) provider based in France, offering a developer-friendly environment for deploying full-stack applications without the need to manage infrastructure manually. CleverCloud was selected based on its strong alignment with the project's core requirements, its compatibility with the selected technology stack, and its ability to ensure full GDPR compliance and data residency within the European Union.

2.1.1 Key Advantages

- **GDPR-Compliant Infrastructure:** CleverCloud hosts applications and databases in data centers located within the European Union (France). This ensures that all user data is processed and stored in compliance with EU data protection regulations, a critical requirement for the PolPaL platform.

- **Developer-Centric Deployment:** The platform supports Git-based deployment, enabling fast and repeatable deployments via a simple git push workflow. Alternatively, for teams using containers, CleverCloud supports Docker image deployment, allowing the application to be developed, tested, and deployed consistently across environments.
- **Full Stack Compatibility:** The PolPaL technology stack-comprising Node.js, Next.js, React, TypeScript, Prisma, and PostgreSQL is fully supported by CleverCloud. Build-packs and custom Docker images can be used to run the application efficiently, whether as a monolith or as decoupled services (e.g., frontend and backend separately).
- **Managed Databases and Services:** CleverCloud provides managed PostgreSQL instances, which integrate seamlessly with Prisma ORM. Automatic backups, performance monitoring, and scaling are included.
- **Secure and Scalable:** CleverCloud includes features such as TLS/SSL certificates by default, horizontal and vertical scaling, log management, and autoscaling for production-readiness. Environment variables and secrets can be securely managed via the web interface or CLI.

2.1.2 Use in Development and Production

During development, the application will be containerized using Docker to ensure consistency between local testing and production deployment. Once tested locally, the Docker container can be automatically deployed to CleverCloud via GitHub. In production, CleverCloud will manage uptime, scaling, and health monitoring, while ensuring that all data remains within EU jurisdictions in compliance with GDPR.

2.2 CleverCloud Pricing Model

CleverCloud offers a transparent and flexible **pay-as-you-go pricing model** designed to accommodate projects of varying sizes, from small prototypes to production-scale applications. The key pricing components are:

- **Containers (Application Runtime):** Applications are billed based on the number and size of running containers (measured in RAM). CleverCloud supports vertical and

horizontal scaling, allowing resources to be adjusted as needed. Container usage is billed per second with monthly caps.

- **Managed Databases:** PostgreSQL and other databases (e.g. Redis, MongoDB) are offered as fully managed services with pricing based on allocated memory and storage. Automatic backups and performance monitoring are included.

All usage is calculated per second. There are no hidden fees, and developers can easily scale up or down depending on project needs. For academic or research projects, this model allows cost-effective and transparent hosting with the flexibility to scale as demand grows.

2.3 Expected Costs

As the project is still ongoing, the exact cost of hosting the web application is an estimate. PaaS often scales costs according to the requirements to run the web application. Upgrading and downgrading is often easy, without the need to set up the whole web application again.

3. Version history

- V1.0 to V1.1: Adding this version control section; otherwise, only editorial changes, mainly proofreading the document and adjusting its layout.