Chronicis - Platform Technical Specification (POC)

Document Version: 2.0
Date: November 18, 2025
Status: Approved for Development
Changelog:

- v2.0 (Nov 18, 2025): Refactored to focus on platform architecture; removed implementation details
- v1.1 (Nov 18, 2025): Added search functionality specifications, rebranded to Chronicis
- v1.0 (Nov 17, 2025): Initial technical specification

Note: Visual design specifications are documented separately in the Chronicis Style Guide. Feature requirements are documented in the Functional Requirements document.

Table of Contents

Platform Overview

Purpose

Chronicis is a web-based knowledge management application for tabletop RPG players to organize campaign information hierarchically.

Platform Type

Single-page application (SPA) with serverless backend

Target Users

- D&D Players (primary)
- Dungeon Masters (future)

Deployment Model

- Cloud-hosted (Azure)
- No local installation required
- Browser-based access

Technology Stack

Frontend

- **Framework:** Blazor WebAssembly (.NET 10)
- **UI Library:** MudBlazor
- **State Management:** Scoped services pattern
- **Browser Requirements:** Modern browsers with WebAssembly support (Chrome, Edge, Firefox, Safari)

Backend

- **API Framework:** Azure Functions (.NET 10)
- **Hosting Model:** Serverless (Azure Static Web Apps managed functions)
- **Authentication:** None for POC (anonymous access)

Data Layer

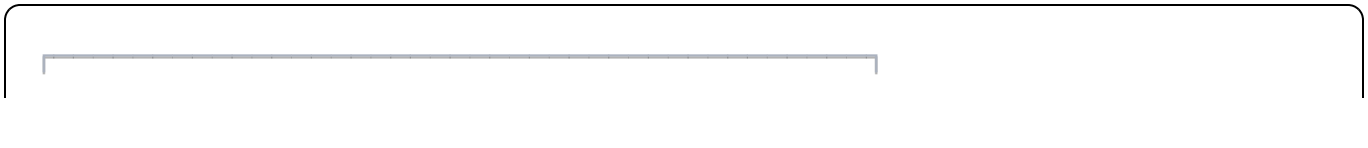
- **ORM:** Entity Framework Core 10
- **Database:** Azure SQL Database
- **Migration Strategy:** Code-first with EF migrations

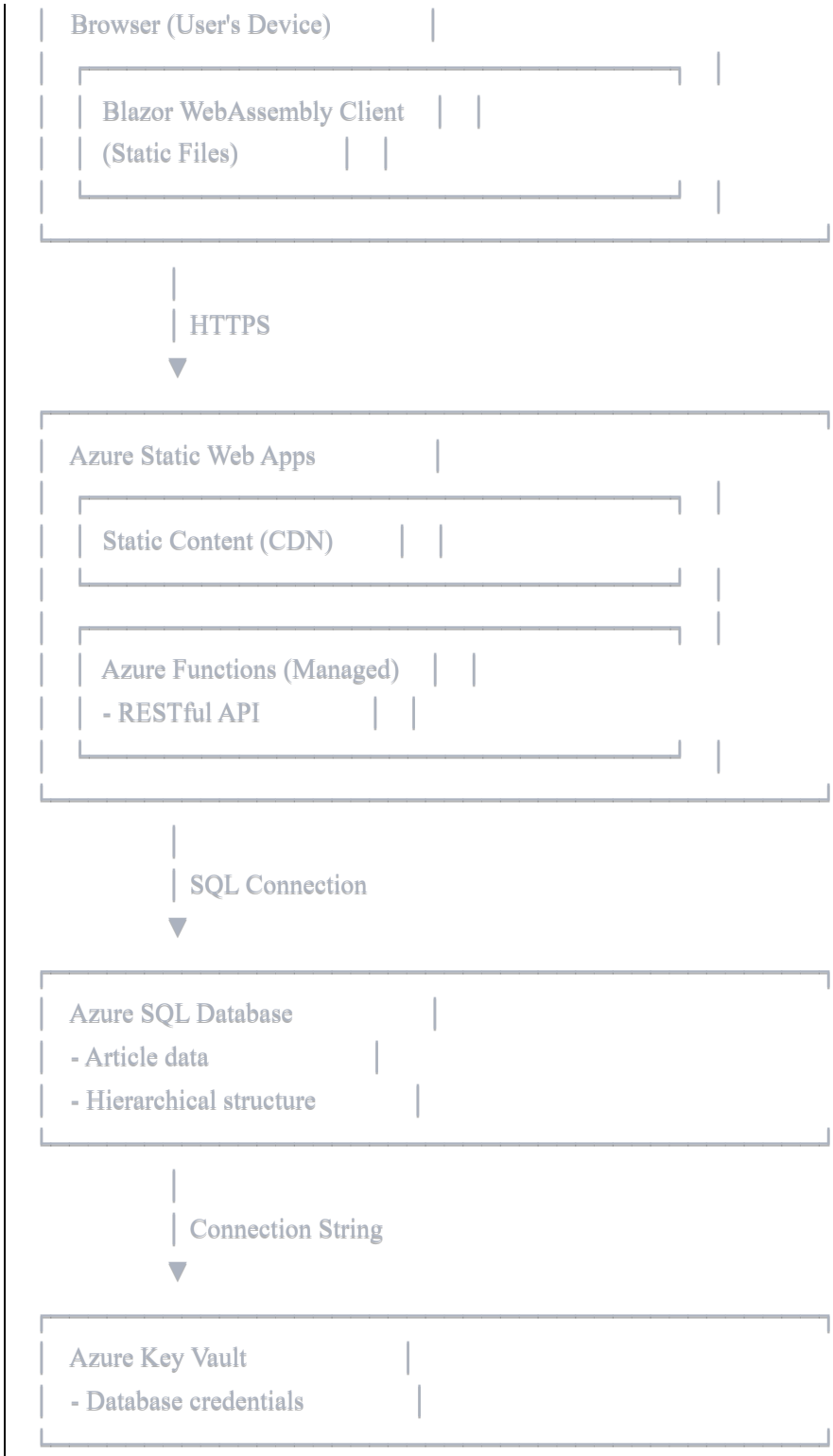
Infrastructure

- **Hosting:** Azure Static Web Apps
- **Database:** Azure SQL Database (Basic or Serverless tier for dev/test)
- **Secrets:** Azure Key Vault
- **CI/CD:** GitHub Actions (auto-configured by Static Web Apps)

Architecture

High-Level Architecture





Communication Patterns

- **Client ↔ API:** HTTP/JSON (RESTful)
- **API ↔ Database:** Entity Framework Core (LINQ)
- **Client State:** In-memory (scoped services)

Scalability Model (POC)

- **Frontend:** Auto-scales via CDN

- **Backend:** Serverless auto-scaling
- **Database:** Single instance (can scale up)

Client Requirements

Core Components

MainLayout:

- App bar with logo and title
- Persistent left drawer (320px width)
- Main content area (flexible width)
- Search box in drawer header

ArticleTreeView:

- Hierarchical tree display
- Lazy loading of children on expand
- Visual distinction for parents (bold text)
- Context menu (three-dot icon on hover)
- Keyboard shortcuts (Ctrl+N for new child)
- Search filtering capability

ArticleDetail:

- Display article title, timestamp, body
- Breadcrumb navigation showing path from root
- Edit and Delete action buttons
- Empty state handling

ArticleEditor:

- Form fields: Title, Date, Time, Body
- Create and Edit modes
- Validation
- Save and Cancel actions

State Management

- TreeStateService: Manages tree data, selection, expansion, search
- ArticleApiService: Handles all API communication
- Error notifications via toast/snackbar

Search Requirements

- Filter articles by title (case-insensitive substring match)
- Display matching articles with full ancestor path
- Trigger: Search button click or Enter key
- Clear: X button in search box
- Auto-expand ancestors of matching articles

Routing

- Single-page with minimal routing
- URL should reflect selected article (optional for POC)

User Interactions

- Click: Select article
- Expand/Collapse: Load/show children
- Context Menu: Add child, Edit, Delete
- Keyboard: Ctrl+N creates child article

Azure Infrastructure

Required Azure Resources

Resource Group:

- Name: `rg-chronicis-dev`
- Location: East US (or preferred region)

Azure Static Web App:

- Name: `swa-chronicis-dev`
- Tier: Free (for POC)
- Includes: Static content hosting + managed Azure Functions

Azure SQL Database:

- Server: `sql-chronicis-dev`
- Database: `Chronicis`
- Tier: Basic or Serverless (dev/test)
- Firewall: Allow Azure services

Azure Key Vault:

- Name: `kv-chronicis-dev`
- Purpose: Store database connection string
- Access: Managed identity from Static Web App

Deployment Pipeline

- GitHub Actions (auto-configured)
- Triggers: Push to main branch
- Steps: Build → Test (optional) → Deploy

Configuration

- Connection string: Stored in Key Vault, referenced in app settings
- CORS: Configured for Static Web Apps domain
- Environment variables: Managed through Azure Portal or GitHub secrets

---

Development Environment

Prerequisites

- .NET 10 SDK
- Azure Functions Core Tools
- Visual Studio 2022 or VS Code with C# extension
- Azure CLI (for infrastructure setup)
- SQL Server (LocalDB, Express, or Docker) for local development

Local Development

- Client runs on `https://localhost:5001` (or configured port)
- API runs on `http://localhost:7071` (Azure Functions runtime)
- Database: Local SQL Server or remote Azure SQL Database

Development Workflow

1. Code changes made locally
2. Test locally with local/remote database
3. Commit to Git
4. Push to GitHub
5. Automatic deployment via GitHub Actions

Database Migrations

- EF Core Code-First migrations
- Apply locally: `dotnet ef database update`
- Apply to Azure: Via CI/CD pipeline or manual script

Non-Functional Requirements

Performance

- **Page Load:** < 3 seconds initial load
- **Tree Expansion:** < 500ms to load children
- **Search:** < 1 second for results
- **Article View:** < 500ms to display

Scalability (POC Scope)

- Support up to 1,000 articles
- Support up to 10 concurrent users
- No real-time collaboration required

Availability

- Target: 99% uptime (POC)
- Scheduled maintenance windows acceptable

Security (POC Scope)

- HTTPS enforced
- No authentication (anonymous access)
- Database credentials in Key Vault
- SQL injection prevention via parameterized queries (EF Core)

Browser Compatibility

- Chrome (latest)
- Edge (latest)
- Firefox (latest)
- Safari (latest)

Data Persistence

- All data persisted to database
- No client-side caching (for POC)
- No offline mode

Error Handling

- User-friendly error messages
- Toast notifications for success/error
- Graceful degradation
- No silent failures

Appendix: Technology Justifications

Why Blazor WebAssembly?

- Full-stack .NET development experience
- Strong typing and compile-time checks
- Component-based architecture
- Good performance for data-heavy UIs
- No server-side rendering needed (lower costs)

Why Azure Functions?

- Serverless = lower costs for POC
- Auto-scaling
- Integrated with Static Web Apps (simplified deployment)
- Pay-per-execution model

Why Azure SQL Database?

- Managed service (no server maintenance)

- Strong consistency for hierarchical data
- Good query performance for recursive queries
- Easy scaling options
- Built-in backup and recovery

Why MudBlazor?

- Comprehensive component library
- Material Design
- Active community
- Free and open source
- Good tree view component

Document History

Version	Date	Author	Changes
2.0	2025-11-18	Technical Team	Refactored to platform-focused specification, removed implementation details
1.1	2025-11-18	Technical Team	Added search functionality, rebranded to Chronicis with dragon logo
1.0	2025-11-17	Technical Team	Initial technical specification based on functional requirements

---

End of Technical Specification