TreveccaPedia

Architectural Manual

Made with arc42

1. Introduction and Goals

TreveccaPedia is a unified information hub for students, faculty, administration, and others to find and contribute information about Trevecca Nazarene University.

Requirements Overview

What is TreveccaPedia?

The main goal of TreveccaPedia is to provide a unified location for all kinds of information about Trevecca. This will be done mainly through Wikis, but also making appropriate use of forums, and possibly other formats, such as discussion boards, galleries, etc.

In order to make this information available and accessible, the project will employ LLM technology, leveraging emerging agentic tooling techniques to integrate AI into the information retrieval and consumption aspect of the site.

TreveccaPedia will also aim to streamline the publication of vital information by official Trevecca departments to students, faculty, etc.

Quality Goals

Nr.	Quality	Motivation
1	*Usability*	*Adoption is highly dependent on convenience and intuitiveness for a target group like ours.*
2	*Authentication*	*Integration with Trevecca SSO for contribution and potentially restricted information.*
3	*Maintainability*	*This codebase will have many different maintainers and developers as students move through the institution.*
4	*Performance*	*Reactivity and responsiveness are vital for adoption and continued use.*
5	*Accessibility*	*Most of the issue with many current information dissemination resources at Trevecca is the barrier to first usage and lack of intuitive interfaces.*

Stakeholders

Role/Name	Expectations
Users	*Easy to access, useful information and intuitive interface. Reliable easy to use platform.*
Students	*Easy to access, useful information and intuitive interface, with useful contribution, collaboration, and discussion resources.*

Role/Name	Expectations
	Private Login w/ SSO
Faculty/Admin	*Unified format and platform to publish and manage information release*.
System Admin	*Thorough moderation and administration tools and dashboards*.

2. Constraints

Technical Constraints

Constraint	Background and/or Motivation
Budget	*Cost effective, open-source, or university licensed tools are preferable to minimize expenses.*
Team size	*Small development team; Must be easy to build and maintain with limited resources;*
Authentication	*University SSO for user verification;*

Organizational Constraints

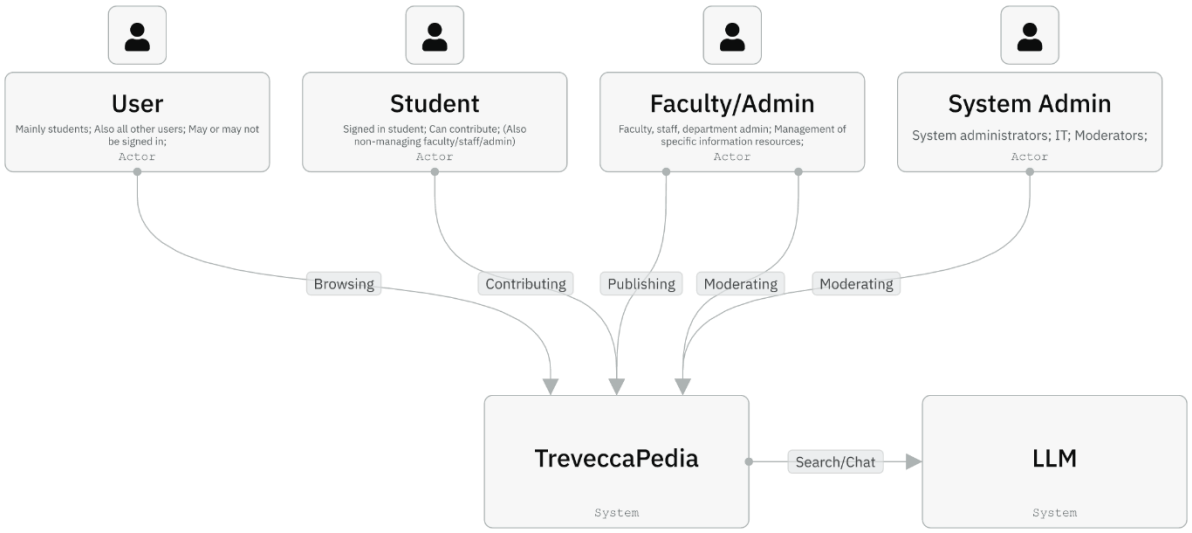
Constraint	Background and/or Motivation
Team	*Small group of students, collaboration will be small scale and low budget (GitHub, Discord, etc.).*
Time Schedule	*Time to market is one academic semester, prioritizing core requirements of the project.*
Development Setup	*Any development configuration must be portable and environment agnostic.*
Configuration/Version Control	*Git for version control/collaborative development.*

Conventional Constraints

Constraint	Background and/or Motivation
Centralized Documentation	*Use of proper documentation for architecture and consistent information that can be easily updated*
Security	*Must ensure necessary security to university standards.*
Accessibility	*Must be easily available to any user at any time.*

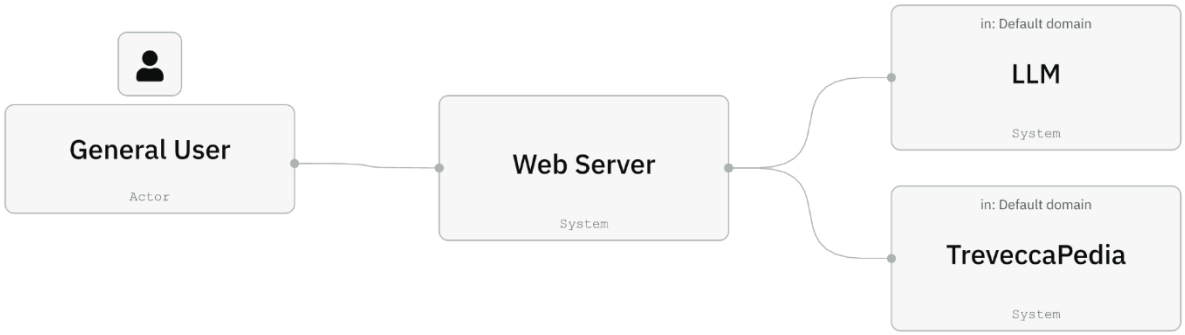
3. Context and Scope

Business Context



Entity	Description
User	*Mainly students; Also all other users; May or may not be signed in;*
Student	*Signed in student; Can contribute; (Also non-managing faculty/staff/admin)*
Faculty/Admin	*Faculty, staff, department admin; Management of specific information resources;*
System Admin	*System administrators; IT; Moderators;*
LLM	*LLM processing for search, summary, chat, etc.;*

Technical Context



Entity	Description
General User	*All users that connect to the web application (includes students, faculty, managers, etc.). Access through Web Server.*
Web Server	*Renders the web page to the User, connecting to TreveccaPedia for data and the LLM for AI results.*
LLM	*Inputs requests from the Web Server, uses MCP tool calls back to the web server to get results from TreveccaPedia for additional context for final result back to Web Server, to be relayed to User.*
TreveccaPedia	*Contains all app data and proper APIs to access and manage it. Also manages business logic and data maintenance.*

4. Solution Strategy

This section summarizes the fundamental architectural decisions and solution approaches shaping the TreveccaPedia system. These decisions support the primary quality goals of Authentication, Usability, and Maintainability.

Architectural Approach

TreveccaPedia follows a service-based architectural style using independently deployable services. Core domains such as Wiki, Forums, Search, AI Integration, and Moderation are treated as separate services with clear boundaries. An API layer coordinates request routing, authentication handling, and policy enforcement without becoming a monolith. This separation enables technology flexibility, easier onboarding for new developers, and independent scalability.

A modern web frontend will consume the API layer. The exact frontend technology has not yet been selected. The decision will be based on usability, accessibility, and long-term maintainability.

The Search Service integrates Large Language Model (LLM) capabilities through an MCP-based orchestration layer. This isolates AI behavior, reduces security risk, and allows future flexibility in choosing self-hosted, cloud-hosted, or third-party LLM providers.

Microsoft SSO is used as the authentication provider. Tokens issued by SSO are validated by the API and passed to services as needed. This reduces friction for users and ensures compliance with university access policies.

Each service manages its own data store. Specific database technologies have not yet been selected; decisions will be based on domain requirements, ease of maintenance, and operational stability. Search and AI functionality will include a separate indexing pipeline to support fast retrieval.

Technology Decisions (Current and Future)

- **Authentication:** Microsoft SSO with role-based authorization.
- **Frontend (TBD):** A modern framework will be selected using criteria such as accessibility support, developer experience, and ease of maintenance.
- **API Layer (TBD):** A stack supporting strong typing, clear contracts, and middleware for authentication will be chosen.
- **Service Data Stores (TBD):** Each service will use the storage technology most appropriate for domain requirements.
- **LLM Integration (TBD):** The strategy for hosting or consuming an LLM will be guided by privacy considerations, cost constraints, and required response times.

Quality Goals and Solution Approaches

Quality Goal	Scenario (Abbrev)	Solution Approach

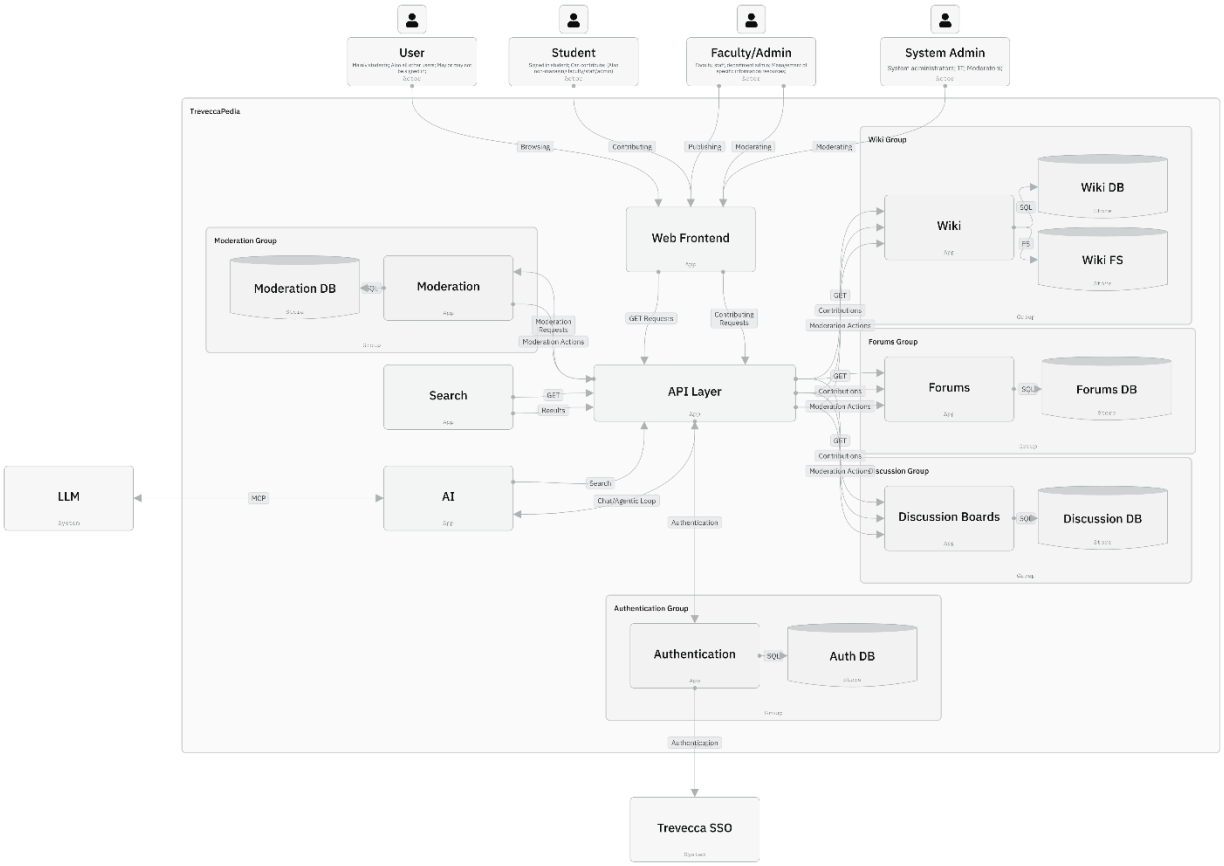
Quality Goal	Scenario (Abbrev)	Solution Approach
Authentication	*Campus users sign in once and access all features*	*Microsoft SSO, token validation, and role-based authorization at API and service level*
Usability	*Students quickly find and contribute information*	*Modern responsive frontend, accessible components, fast search, clear information architecture*
Maintainability	*New contributors can add features safely*	*Independent services, clear API contracts, strong typing, CI enforcement, ADRs*
Performance	*Users receive fast search and AI responses*	*Caching, retrieval-based search, asynchronous pipelines, streaming responses*

Organizational Decisions

The team will use an Agile workflow. Each service will have a designated ownership group responsible for code, deployment, and monitoring. The IT department will be consulted early regarding SSO and data privacy requirements.

5. Building Block View

Whitebox Overall System



Motivation

To portray the high-level, general flow of data through the entire system, as well as the interactions with the main external systems and parties.

Contained Building Blocks

Black box	Responsibilities
User, Student, Faculty/Admin, System Admin	*Users that interact with the system.*
API Layer	*Central entry point, routing, rate-limiting, request aggregation*
Web Frontend	*Single-Page Application (SPA) used by all user roles for browsing, editing, chatting, searching, moderating*
Wiki Service (etc.)	*Article creation, versioning, rendering, history, categories*

Black box	Responsibilities
Moderation Service	*Review queues, reports, bans, content flags, moderation dashboards*
Search Service	*Full text + semantic search across all content sources*
AI Service	*Overview, summarization, natural language interaction, and agentic loops, powered by external LLM*
Authentication Service	*SSO integration, authentication, user database management, roles/permissions management*
LLM	*Large Language Model, controlled via MCP*
Trevecca SSO	*Institutional authentication and user-management system*

Important Interfaces

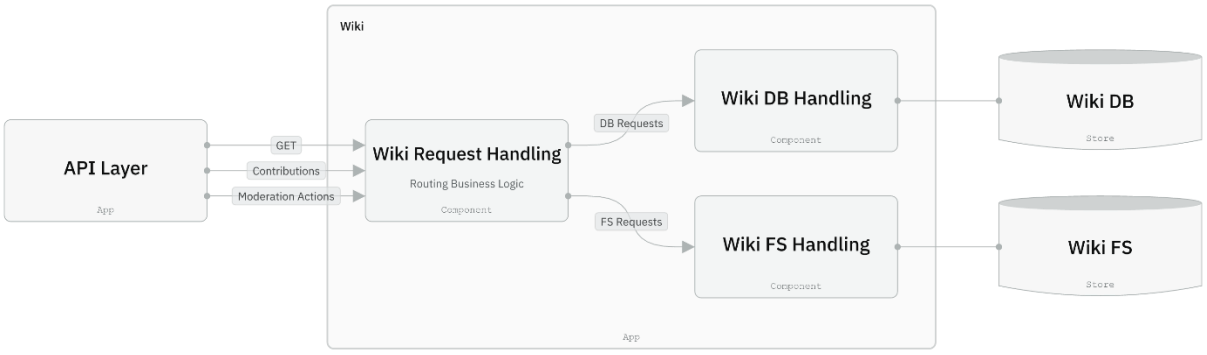
Interface	Description
API Layer	*All internal communication between services goes through the API layer.*
AI Service ⇔ LLM	*Model context protocol (MCP) standards for agentic loops and responses.*

Level 2

Whitebox – *API Layer*

Intakes requests, routes them to the proper service and returns responses.

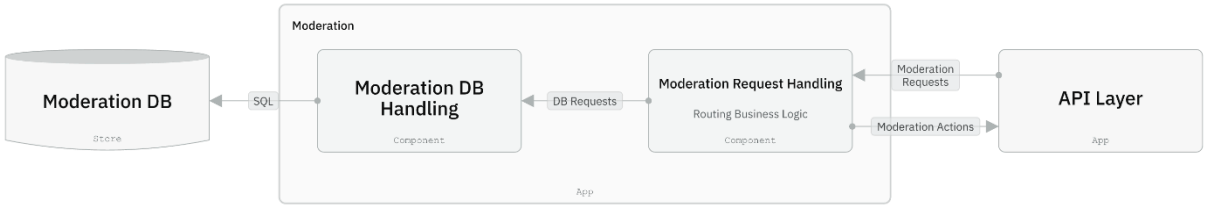
Whitebox – *Wiki Service*



Black Box	Responsibility
Request Handling	*Receives requests from the API layer, handles business logic, and routes requests and responses to and from the*

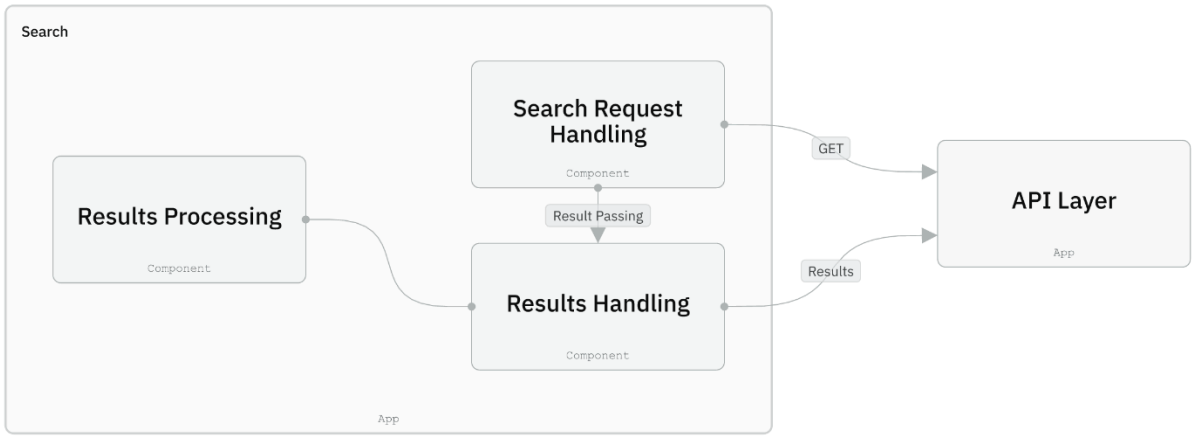
Black Box	Responsibility
	other services.
Database Handling	*Interacts with the Wiki DB (SQL) to perform CRUD actions.*
File System Handling	*Interacts with the File System to perform CRUD actions.*

Whitebox – *Moderation Service*



Black Box	Responsibility
Request Handling	*Receives requests from the API layer, handles business logic, routes requests and responses to and from other services. Initiates requested moderation actions through API Layer.*
Database Handling	*Interacts with the Moderation DB (SQL) to perform CRUD actions.*

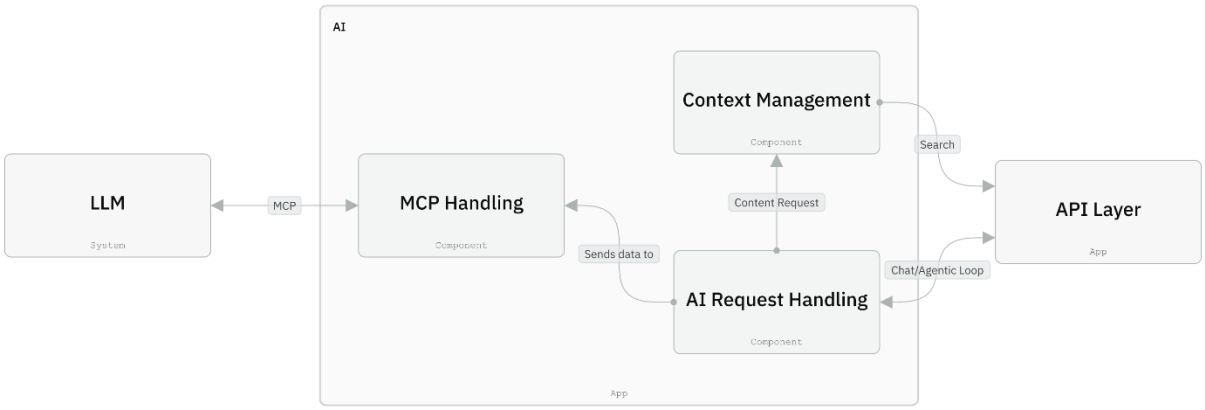
Whitebox – *Search Service*



Black Box	Responsibility
Search Request Handling	*Handles business logic of incoming requests, requests content from content services.*

Black Box	Responsibility
Results Handling	*Handles results, formatting, etc.; returns results.*
Results Processing	*Sorts and organizes results.*

Whitebox - *AI Service*



Black Box	Responsibility
AI Request Handling	Routes AI requests, requests additional context/search results, handles WebSockets.
Context Management	*Requests results from Search Service, formats for use as context in LLM interaction.*
MCP Handling	*Formats and performs MCP requests/loops with LLM.*

6. Runtime View

The following runtime scenarios illustrate the most architecturally significant behaviors of the system. They demonstrate how the system's building blocks interact during key operations.

6.1 Scenario: User Retrieves Information via AI Search or Chat

1. The user enters a question into the AI Chat or Search interface.
2. The frontend sends the request and optional authentication token to the API layer.
3. The API validates the token and forwards the request to the AI Service.
4. The AI Service retrieves relevant content through the Search Service using keyword search and vector-based retrieval.
5. The Search Service sends a structured prompt to the AI Service, which invokes the LLM using MCP.
6. The LLM processes the request and returns a draft answer.
7. The AI Service applies safety checks and assembles the final result.
8. The API streams the result to the frontend.
9. The frontend displays the answer and any supporting citations.

6.2 Scenario: User Contributes to the Wiki

1. A signed-in user opens a wiki page and makes an edit or creates a new page.
2. The frontend sends the request with the user's authentication token to the API.
3. The API validates the user's role and forwards the request to the Wiki Service.
4. The Wiki Service stores the new version and keeps version history.
5. The Wiki Service emits an event indicating that the page was updated.
6. The Search/Indexing process asynchronously re-indexes the updated content.
7. The frontend displays a confirmation and updates the page view.

6.3 Scenario: User Posts a Thread in the Forums

1. A signed-in user writes a new forum post or thread.
2. The frontend sends the request and token to the API.
3. The API verifies permissions and forwards the request to the Forums Service.
4. The Forums Service stores the new thread in its data store.
5. The Moderation Service evaluates the content automatically and may flag it for review.
6. The frontend displays the new thread to the user.

6.4 Scenario: Admin Moderates Content

1. An admin opens the moderation dashboard.
2. The API verifies that the user has elevated permissions.

3. The Moderation Service returns a queue of flagged or reported items.
4. The admin reviews the items and takes actions such as approve, edit, mask, or remove.
5. When content is modified or removed, the relevant service (Wiki or Forums) applies the change.
6. Search/Indexing re-indexes any modified content.
7. All moderation decisions are logged for audit purposes.

6.5 Scenario: System Administrator Handles Outdated or Incorrect Information

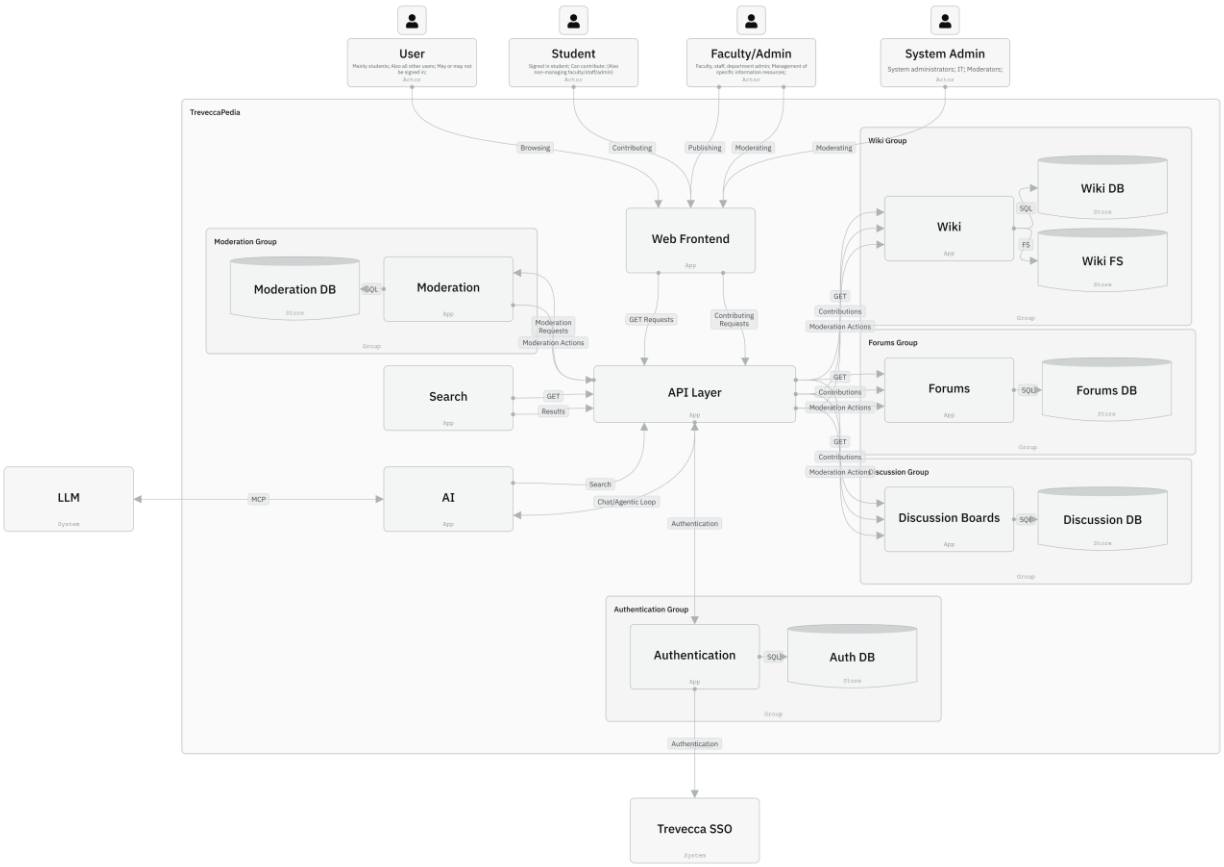
1. The System Admin generates a report of potentially outdated or high-traffic, low-quality content.
2. The Wiki and Forums Services provide lists of candidates based on metadata such as last modified date and user feedback.
3. The Search Service indicates how often outdated pages appear in search results.
4. The System Admin triggers re-indexing tasks or assigns review actions to content owners.
5. Critical content may be temporarily demoted or labeled until updates are completed.

7. Deployment View

TreveccaPedia is deployed across multiple logical nodes: a Web Frontend, an API Layer, several domain-specific backend services (Wiki, Forums, Discussion Boards, Search, Moderation, Authentication), multiple data stores, an external Trevecca SSO system, and an external LLM provider.

These nodes are independent, communicate through API calls, and can be scaled or deployed separately.

Infrastructure Level 1



TreveccaPedia runs in a multi-node web architecture that contains the following major infrastructure elements:

- **Client Browser** – Any user device (desktop, laptop, mobile) accessing TreveccaPedia through the Web Frontend.
- **Web Frontend** – Hosts the user-facing web application. Serves UI assets and forwards user requests to the API Layer.

- **API Layer** – A centralized gateway for coordinating requests between frontend and backend services.
- **Backend Service Nodes** – Independently deployable services aligned with the App Groups. These include:
    - Wiki Service
    - Forums Service
    - Discussion Boards Service
    - Search Service
    - AI Service
    - Moderation Service
    - Authentication Service
- **Data Stores** – Separate databases and file storage for each domain area:
    - Auth DB
    - Wiki DB + Wiki File Storage (FS)
    - Forums DB
    - Discussion DB
    - Moderation DB
- **Trevecca SSO** – External identity provider for authentication.
- **LLM Provider** – External system used by the AI and Search services through MCP-based request handling.
- **Network / Internet** – Communication channels between all nodes.

Motivation

TreveccaPedia is intended to be built as a system of **independently deployable services**, allowing each domain area (Wiki, Forums, Search, etc.) to evolve separately, be maintained by different student teams, and scaled independently based on usage.

Separating databases prevents cross-service coupling and keeps data access patterns clean and domain-specific.

Using Trevecca SSO allows the project to integrate directly with the university's authentication ecosystem.

Leveraging an external LLM reduces infrastructure complexity while allowing the Search and AI components to use MCP-based context requests.

Mapping of building blocks to infrastructure

- **Web Frontend** → Client + Frontend Node
- **API Layer** → API Node
- **Authentication App** → Authentication Service Node + Auth DB
- **Wiki App** → Wiki Service Node + Wiki DB + Wiki File Storage
- **Forums App** → Forums Service Node + Forums DB
- **Discussion Boards App** → Discussion Service Node + Discussion DB
- **Moderation App** → Moderation Service Node + Moderation DB
- **Search App** → Search Service Node
- **AI App** → AI Service Node (communicates with external LLM)
- **External Trevecca SSO** → Identity Provider Node
- **External LLM Provider** → AI Processing Node

Each service communicates with the API Layer, which acts as the central orchestrator for user-facing operations.

Infrastructure Level 2

7.2.1 API Layer Node

The API Layer hosts routing and orchestration logic. It handles incoming HTTP requests from the Web Frontend and delegates work to services such as Wiki, Forums, Search, Moderation, etc. It also coordinates AI/MCP requests.

7.2.2 Web Frontend Node

Contains static UI assets, authentication of redirect logic (SSO), and code for initiating requests to the API Layer. It does not contain business logic.

7.2.3 Wiki Service Node

Contains:

- Wiki Request Handling component
- Wiki DB Handling component
- Wiki FS Handling component

Communicates with:

- Wiki DB

- Wiki FS
- API Layer

7.2.4 Search Service Node

Contains:

- Search Request Handling
- Results Processing
- Results Handling

Communicates with:

- API Layer
- AI Service (for MCP-based processing when necessary)

7.2.5 Moderation Service Node

Contains:

- Moderation Request Handling
- Moderation DB Handling

Communicates with:

- Moderation DB
- API Layer

7.2.6 AI Service Node

Contains:

- AI Request Handling
- Context Management
- MCP Handling

Communicates with:

- External LLM provider
- API Layer
- Search Service (for enhanced search context)

7.2.7 Authentication Service Node

Contains:

- Authentication logic
- Connections to Auth DB

Communicates with:

- Trevecca SSO
- API Layer

8. Cross-cutting Concepts

Dependencies between the modules

Loosely coupled to allow for replacements. All major subsystems: Web Frontend, API Layer, and functional/business Services communicate with well-defined interfaces.

API Layer validates incoming requests, forwarding calls to internal service (User, forum, etc.).

REST alignment provides predictable behavior with defined requests for each service and clear responsibilities. Those services can be altered without breaking existing services if needed.

User Interface

Interactions- Login via SSO redirecting to our UI, browsing wiki pages, viewing posts, using LLM for Searches.

Design and framework TBD

Single-Page Application: Web Frontend maintains the navigation and interaction of all modules. UI dynamically adjusts to authenticated user roles without reloading pages.

WebSocket Integrations: Real time updates for AI Search, moderation, and editing. Ensuring users transfer information between the modules with uniform implementation.

Validation

API Layer- ensures requests are valid for to sign in, authentication, and user role permissions.

Each Layer consists of individual validation with SSO as the foundational source of truth. All users and their roles are validated using SSO tokens, and managed by the Authentication Service.

Services validate the content and database operation with user role permissions and content appropriate rules.

Exceptions and Error Handling

Internal services handle expectations; API layer only handles requests and responses. Failures are handled independently through a RESTful API. The Web frontend can display user-friendly messages. This allows the system to continue normally, even in the case of unforeseen circumstances and behavior.

Robust logging will be critical to maintenance and improvement of the system in the future.

9. Architecture Decisions

This section documents the **major architectural decisions**, the alternatives considered, and the reasoning behind each choice. Each decision is written as an **ADR (Architecture Decision Record)** following the Nygard structure.

ADR 1 — Service-Based Architecture with Independently Deployable Services

Context
TreveccaPedia will provide multiple features (Wiki, Forums, AI Search/Chat, Moderation). These have different responsibilities and may evolve independently. The system must support maintainability, scalability, and clear separation of concerns. Team members will change over time as students graduate, so a structure that reduces coupling is beneficial.

Decision
Adopt a **service-based architecture** with **independently deployable services** (Wiki Service, Forum Service, Search Service, Moderation Service, Authentication boundary).

Status
Accepted

Consequences

- **Positive:**
  - Each service can be developed, tested, and deployed independently.
  - Improves maintainability as teams can work on different services without conflicts.
  - Enables scaling specific services (e.g., Search/AI) without scaling the whole system.
- **Negative:**
  - Requires more infrastructure (APIs, networking, load balancing).
  - Requires monitoring, logs, and observability across services.
- **Neutral:**
  - Increases architectural complexity, but manageable for long-term growth.

ADR 2 — Use of Microsoft Single Sign-On (SSO) for Authentication

Context
Trevecca University currently uses Microsoft products for identity management. Students, faculty, and staff already authenticate using Microsoft credentials. Authentication is a top

priority (Quality Goals #2). Privacy and account integrity require integration with an official provider.

Decision
Use **Microsoft SSO** as the authentication provider for all authenticated features of TreveccaPedia.

Status
Accepted

Consequences

- **Positive:**
  - Matches existing campus identity workflows.
  - Reduces security risks by delegating authentication to Microsoft.
  - Simplifies user management since accounts already exist.
- **Negative:**
  - Adds dependency on Trevecca's IT infrastructure.
  - Requires careful integration with backend services.
  - Limits custom authentication flows.
- **Neutral:**
  - Some parts of the application may remain publicly accessible.

ADR 3 — AI Search Implemented Using MCP (Model Context Protocol)

Context
The system requires AI-powered search and chat. The team wants the LLM to access internal TreveccaPedia data safely and repeatedly. MCP (Model Context Protocol) is designed for tool-based interactions between an LLM and external systems.

Decision
Implement all LLM interactions through **MCP**, where the Search Service exposes tool calls that the LLM can use to retrieve wiki and forum data.

Status
Accepted (pending final selection of the LLM provider)

Consequences

- **Positive:**
  - Clear and safe communication channel between LLM and TreveccaPedia.
  - Enables controlled access to internal data via tools instead of free-form scraping.

    - Makes search results more accurate and personalized.
- **Negative:**
    - Requires careful design of tools and endpoints.
    - LLM cost and performance may vary depending on the provider.
- **Neutral:**
    - Does not lock the team into a specific LLM yet.

ADR 4 — Web API as Central Gateway Between Frontend, Services, and LLM

Context
The system needs a clear communication structure between front-end apps and back-end services. A single point of orchestration simplifies request handling and ensures consistent authentication and authorization.

Decision
Use a **central API Layer** as the main gateway for all traffic between frontend, backend services, and MCP/LLM.

Status
Accepted

Consequences

- **Positive:**
    - Consistent auth checks and session validation.
    - Clear logging, monitoring, and throttling at a single entry point.
    - Simplifies front-end logic.
- **Negative:**
    - The API gateway may become a bottleneck if not designed for scalability.
    - Requires routing logic and service discovery.
- **Neutral:**
    - Does not dictate the final API technology (Node, Python, Go, etc.).

ADR 5 — Use of a Relational Database (Tentative) for Core Data

Context
Wiki pages, forums, moderation logs, and user contributions all require structured, relational data with clear ownership and auditability. Because the specific technology is not yet chosen, the team identified *requirements* but not a vendor.

Decision
Use a **relational database** for core TreveccaPedia data (likely MySQL, PostgreSQL, or SQL Server).

Status
Proposed

Consequences

- **Positive:**
  - Strong data consistency.
  - Easy to track relationships (users → contributions).
  - Works well with versioning and audit trails.
- **Negative:**
  - Requires schema design and migration management.
  - Might need horizontal scaling solutions later.
- **Neutral:**
  - Final DB technology selection will be captured in a future ADR.

ADR 6 — UI / Frontend Stack

Context:
We haven't selected exact front-end technologies yet. However, usability, accessibility, and maintainability are top quality goals. The front end must support SSO, rich text editing, and AI features.

Decision:
Adopt a modern **component-based web framework** such as **React**, **Vue**, or **Next.js** (decision deferred until prototype phase). Requirements:

- Strong community support
- Easy SSO integration
- Support for reusable UI components
- Good accessibility patterns built-in

Status:
Proposed (final selection pending).

Consequences:

- **Positive:** Modern tooling improves usability and frontend maintainability.
- **Negative:** Learning curve for contributors unfamiliar with chosen framework.
- **Neutral:** Final choice depends on skill sets of the team.

10. Quality Requirements

Quality Tree

- Usability
    - Learnability – U01
    - Operability – U02
- Authentication
    - Secure institutional SSO integration – A01
    - Role-based access control – A02
- Maintainability
    - Modularity – M01
    - Analyzability & testability – M02, M03
    - Independent deployability & scalability – M01
- Performance
    - Load times – P01
    - Reactivity – P02
- Accessibility
    - Operability – A01
    - Understandability – A01
    - Robustness – A01

Quality Scenarios

ID	Quality	Scenario
U01	Usability	New user can successfully edit and save a wiki page in < 5 minutes.
U02	Usability	Moderator can remove inappropriate content and confirm action in ≤ 3 minutes.
A01	Authentication	Institutional SSO login completes with correct role assignment in ≤ 3 seconds.

ID	Quality	Scenario
A02	Authentication	Any unauthorized action is blocked 100% of the time with immediate audit log.
M01	Maintainability	A single service can be deployed to production with zero downtime in ≤ 5 minutes
M02	Maintainability	Logs are robust and provide robust analytical data.
M03	Maintainability	Adding a small feature, bug fix, or rule change can be done by any developer in a reasonable amount of time.
P01	Performance	Less than one second to initial page load on any given page.
P02	Performance	Majority of buttons have a response in less than 500ms.
A01	Accessibility	Mobile web app is fully functional and displays well.

11. Risks and Technical Debts

Technical Risks

MCP & LLM Available

The search feature depends on the external LLM. If either the MCP tool call or LLM is unavailable or slow, the subsystem can degrade. If a third-party LLM is used, data security may bring additional risk considerations.

Impacts:

> *Search is slow or empty, Users get no response*
>
> ***Severity:*** *High*

SSO Failures

Access depends on SSO. I.e. if it were down or breaks, users cannot login or access restricted content.

Impacts:

> *Many parts of the app become inaccessible to users, rendering the website dysfunctional*
>
> ***Severity:*** *High*

Inconsistency between layers

API and services could be ruled as accepted, but data could be invalid or rejected.

Severity: Medium

State Failure

Updates, indexing, and posts are not properly written/updated.

Impacts: *Partial writing/forums; No content shown*

12. Glossary

Term	Definition
LLM	*Large Language Model*
MCP	*Model Context Protocol*
SSO	*Single Sign-On (Trevecca institutional Microsoft SSO)*
Wiki	*organization of collaborative, informative pages*