Technology Choices

Our system architecture is based on a modular microservices design, where each service is independently deployed and maintained. We carefully evaluated at least three technology options for each major system component before finalizing our stack.

Frontend Framework

Option	Overview	Pros	Cons
React (Chosen)	Declarative JS library for building UIs	Large ecosystem, reusable components, strong community, PWA support	Requires setup, fast-moving ecosystem, JSX learning curve
Angular	Full-featured frontend framework	Built-in features, strict structure, TypeScript native	Heavier, steeper learning curve
Vue.js	Progressive framework with a gentle learning curve	Simple syntax, great docs, good performance	Smaller ecosystem, fewer integrations for large-scale systems

Justification: React provides component modularity and mature support for PWAs, aligning with our need for a responsive, maintainable interface that integrates seamlessly with our microservices. Its ecosystem also allowed us to rapidly implement custom search features and theme support.

Backend Framework (API Layer)

Option	Overview	Pros	Cons
FastAPI (Chosen)	Python-based high-performance async API framework	Async support, OpenAPI docs built-in, very fast, type hints supported	Newer ecosystem, fewer extensions than Django
Flask	Lightweight Python web framework	Simple and flexible, large community	No built-in async support, requires

			more boilerplate
Django	Full-featured Python web framework	ORM, admin interface, built-in features	Monolithic, harder to separate into microservices cleanly

Justification: FastAPI gives us performance and async support critical for concurrent services like search and analytics. Its automatic doc generation and use of type hints accelerate development and reduce bugs.

Backend Language

Option	Overview	Pros	Cons
Python (Chosen)	General-purpose, dynamically typed language	Readable, productive, mature ML/NLP libraries	Slower performance compared to compiled languages
Go	Compiled systems language	Very fast, great for concurrency and microservices	Verbose, smaller ecosystem for data processing
Node.js (JS)	JS runtime for server-side code	Unified frontend/backend language, async by default	Callback complexity, inconsistent async patterns

Justification: Python balances speed of development with ecosystem richness. Since we use NLP for AI search (via spaCy), Python is ideal.

Containerization

Option	Overview	Pros	Cons
Docker (Chosen)	Standardized container runtime	Consistent across environments, easy local setup	Requires Docker knowledge
Vagrant	Virtualization tool	Full OS-level control	Heavier, slower, deprecated
Bare Metal	Manual dependency management	Full control	Hard to replicate, error-prone

Justification: Docker allows each microservice to run in a consistent, isolated environment. It enables local development to match production deployments and simplifies team onboarding.

Summary

Our technology stack: React, FastAPI with Python, and Docker was chosen for its strong modularity, ease of maintenance, and developer productivity. React enables the creation of responsive, user-friendly interfaces that function seamlessly across all device types. FastAPI offers fast development with built-in validation and documentation, making it ideal for building clean and maintainable APIs. Docker ensures consistent environments across development and production, supporting a scalable microservices architecture. These choices align directly with our goals of building an offline-capable, user-centered platform that is simple to deploy, easy to extend, and reliable across a wide range of usage contexts.