VYGA - ADR FOR KEY DECISIONS

ADR 001: Selection of Backend Framework

Context

The backend of the platform requires a flexible, lightweight framework to develop RESTful APIs that can handle user authentication, content aggregation, search functionalities, and AI-supported processes.

Decision

The backend will be developed using Python with the Flask framework.

Rationale

- **Flexibility**: Flask is a micro-framework that allows for easy customization and is not opinionated, meaning it doesn't enforce any particular way of structuring the application.
- **Community Support**: Flask has a large and active community, providing plenty of resources, plugins, and third-party libraries.
- **Integration**: Python's extensive ecosystem, including libraries like Pandas, NumPy, and TensorFlow/PyTorch, can be easily integrated within Flask applications.

Consequences

- **Scalability**: Flask's lightweight nature might require additional configuration and management for larger-scale applications.
- **Learning Curve**: Developers familiar with Python will find Flask easy to learn and use, reducing the ramp-up time for new developers joining the project.

Status

Approved

ADR 002: Selection of Frontend Framework

Context

The frontend of the platform needs to be dynamic and responsive, capable of interacting efficiently with backend APIs to provide real-time content updates and search capabilities.

Decision

The frontend will be developed using React.

Rationale

- **Dynamic UI**: React's component-based architecture allows for the creation of reusable UI components, which enhances development speed and maintainability.
- **SPA**: React is well-suited for developing Single Page Applications (SPAs), providing a seamless user experience without full-page reloads.
- **Community and Ecosystem**: React has a robust ecosystem and extensive community support, with numerous libraries and tools available for enhancing functionality.

Consequences

- **SEO Challenges**: React SPAs may face issues with search engine optimization (SEO); however, this can be mitigated with server-side rendering (SSR) or pre-rendering techniques.
- **Complexity**: While React offers powerful tools, it also introduces a level of complexity, especially with state management, which may require additional libraries like Redux.

Status

Approved

ADR 003: Database Management Solution

Context

The platform requires a database solution that can handle large volumes of unstructured data with high performance, availability, and scalability.

Decision

MongoDB Atlas will be used for managing the database.

Rationale

- **NoSQL Flexibility**: MongoDB's document-based structure is ideal for storing unstructured or semi-structured data.
- **Scalability**: MongoDB Atlas provides automated scaling options, which is essential for a growing platform.
- **Managed Service**: Using MongoDB Atlas reduces operational overhead by offering a fully managed cloud service with built-in security and backup options.

Consequences

- **Data Modeling**: The flexibility of NoSQL can lead to challenges in data modeling and may require careful planning to avoid performance issues.
- **Cost**: While MongoDB Atlas provides convenience, it may introduce additional costs compared to self-managed solutions.

Status

Approved

ADR 004: AI Model Training and Deployment Framework

Context

The platform needs to incorporate AI models for content curation and search optimization, requiring a robust framework for model training and deployment.

Decision

AI models will be developed and deployed using TensorFlow or PyTorch.

Rationale

- **Flexibility**: Both TensorFlow and PyTorch offer powerful tools for developing and training machine learning models, with TensorFlow being more production-oriented and PyTorch more research-oriented.
- **Ecosystem**: Both frameworks have a rich ecosystem of tools, libraries, and community support, facilitating model development and deployment.
- **Scalability**: These frameworks are designed to work well with cloud platforms like AWS, ensuring that models can scale with the platform's needs.

Consequences

- **Complexity**: The complexity of AI models and their deployment might require specialized knowledge, which could impact development timelines.
- **Resource Intensive**: Training and deploying AI models can be resource-intensive, requiring significant computational power and potentially increasing infrastructure costs.

Status

Approved

ADR 005: Containerization Strategy

Context

The platform needs to ensure consistent environments across development, testing, and production, along with easy scaling and deployment of services.

Decision

Docker will be used for containerizing applications.

Rationale

- **Environment Consistency**: Docker ensures that the application runs the same way across different environments, reducing the “it works on my machine” problem.
- **Scalability**: Docker containers can be easily scaled horizontally to meet the growing demand.
- **Portability**: Docker containers can be deployed on any system that supports Docker, providing flexibility in choosing deployment environments.

Consequences

- **Learning Curve**: Developers need to be familiar with containerization concepts and Docker, which may require training.
- **Resource Overhead**: Running multiple containers can introduce resource overhead, which needs to be managed, especially in resource-constrained environments.

Status

Approved

ADR 006: Orchestration and Container Management

Context

The platform requires a solution for orchestrating and managing containerized applications in a scalable and secure manner.

Decision

AWS Elastic Container Service (ECS) will be used for orchestration.

Rationale

- **Managed Service**: AWS ECS is a fully managed service, reducing the operational burden of managing the container orchestration infrastructure.
- **Integration**: ECS integrates seamlessly with other AWS services like Elastic Container Registry (ECR) and AWS IAM for secure and efficient operations.
- **Scalability**: ECS offers robust scaling options, enabling the platform to handle varying loads efficiently.

Consequences

- **Vendor Lock-in**: Using AWS ECS may lead to vendor lock-in, making it challenging to migrate to another platform in the future.
- **Cost**: While ECS simplifies container management, it may incur higher costs compared to self-managed solutions.

Status

Approved