# Detailed Database Recommendation for LangGraph Designer in Azure

Based on my analysis of the codebase, I'll provide a comprehensive recommendation for implementing a database solution in Azure for the LangGraph Designer application.

## Current State Analysis

The application currently uses in-memory storage with Zustand for state management. All data is lost when the application is refreshed or closed, including:

1. **Workflow Definitions** - Agents, tools, memory nodes, operators, and their connections
2. **Work Groups** - User access management and permissions
3. **User Data** - Authentication and authorization information
4. **Version History** - Workflow and component versioning

## Recommended Database: Azure Cosmos DB

I recommend **Azure Cosmos DB** with the **SQL API** for the following reasons:

1. **Flexible Schema** - The application has complex nested data structures with varying properties based on node types, which fits well with a document database
2. **Scalability** - Can scale to handle large numbers of workflows and users
3. **Global Distribution** - Provides low-latency access worldwide
4. **Multi-model Support** - SQL API provides familiar query syntax while maintaining document flexibility
5. **Versioning Support** - Built-in support for versioning through TTL and change feed
6. **Azure Integration** - Seamless integration with other Azure services like Azure Functions, API Management, and Azure AD

## Database Structure

Here's a detailed breakdown of the database collections and their schemas:

### 1. Users Collection

json

CopyInsert

{

"id": "string",

"email": "string",

"name": "string",

"role": "string",

"createdAt": "timestamp",

"lastLogin": "timestamp",

"preferences": {

"theme": "string",

"defaultView": "string"

}

}

### 2. WorkGroups Collection

json

CopyInsert

{

"id": "string",

"name": "string",

"owner": "string (userId)",

"scope": "string (Public/Restricted)",

"description": "string",

"createdAt": "timestamp",

"updatedAt": "timestamp"

}

### 3. WorkGroupUsers Collection

json

CopyInsert

{

"id": "string",

"workGroupId": "string",

"userId": "string",

"access": "string (Admin/Editor/Viewer)",

"validationDaysRemaining": "number",

"lastValidated": "timestamp",

"entityRolePairs": [

{

"entity": "string",

"role": "string"

}

]

}

### 4. AccessRequests Collection

json

CopyInsert

{

"id": "string",

"workGroupId": "string",

"requestorId": "string",

"requestedDate": "timestamp",

"status": "string (Pending/Approved/Rejected)",

"accessType": "string (partial/admin)",

"entityRolePairs": [

{

"entity": "string",

"role": "string"

}

],

"reviewedBy": "string",

"reviewedDate": "timestamp"

}

### 5. Workflows Collection

json

CopyInsert

{

"id": "string",

"name": "string",

"description": "string",

"workGroupId": "string",

"createdBy": "string (userId)",

"createdAt": "timestamp",

"updatedAt": "timestamp",

"version": "string",

"versionedId": "string",

"isPublished": "boolean",

"tags": ["string"]

}

### 6. WorkflowVersions Collection

json

CopyInsert

{

"id": "string",

"workflowId": "string",

"version": "string",

"versionedId": "string",

"createdBy": "string (userId)",

"createdAt": "timestamp",

"nodes": [

{

"id": "string",

"type": "string",

"name": "string",

"content": "string",

"position": { "x": "number", "y": "number" },

"llmModel": "string",

"memoryType": "string",

"toolType": "string",

"operatorType": "string",

"parentId": "string",

"icon": "string",

"operatorConfig": {}

}

],

"edges": [

{

"id": "string",

"source": "string",

"target": "string",

"sourceHandle": "string",

"targetHandle": "string",

"animated": "boolean"

}

]

}

### 7. ToolLibrary Collection

json

CopyInsert

{

"id": "string",

"name": "string",

"description": "string",

"type": "string",

"code": "string",

"version": "string",

"versionedId": "string",

"createdBy": "string (userId)",

"createdAt": "timestamp",

"updatedAt": "timestamp",

"tags": ["string"],

"isPublic": "boolean",

"workGroupId": "string"

}

### 8. Deployments Collection

json

CopyInsert

{

"id": "string",

"workflowId": "string",

"workflowVersionId": "string",

"environment": "string (dev/staging/prod)",

"status": "string (active/inactive/failed)",

"deployedBy": "string (userId)",

"deployedAt": "timestamp",

"lastExecuted": "timestamp",

"executionCount": "number",

"endpoint": "string"

}

### 9. ExecutionLogs Collection

json

CopyInsert

{

"id": "string",

"deploymentId": "string",

"workflowId": "string",

"startTime": "timestamp",

"endTime": "timestamp",

"status": "string (success/failure/in-progress)",

"input": "object",

"output": "object",

"nodeExecutions": [

{

"nodeId": "string",

"startTime": "timestamp",

"endTime": "timestamp",

"status": "string",

"input": "object",

"output": "object",

"error": "string"

}

],

"error": "string"

}

## Database Creation Prompt for Agentic IDE

CopyInsert

Create an Azure Cosmos DB database for the LangGraph Designer application with the following specifications:

1. Create a new Azure Cosmos DB account with the SQL API

2. Database name: LangGraphDesignerDB

3. Create the following collections with appropriate partition keys:

a. Users Collection

- Partition key: /id

- Indexing: Default

- TTL: None

- Unique keys: /email

b. WorkGroups Collection

- Partition key: /id

- Indexing: Default

- TTL: None

- Unique keys: /name

c. WorkGroupUsers Collection

- Partition key: /workGroupId

- Indexing: Default

- TTL: None

- Unique keys: Composite (/workGroupId, /userId)

d. AccessRequests Collection

- Partition key: /workGroupId

- Indexing: Default

- TTL: 90 days (for rejected requests)

e. Workflows Collection

- Partition key: /workGroupId

- Indexing: Default

- TTL: None

- Unique keys: Composite (/workGroupId, /name)

f. WorkflowVersions Collection

- Partition key: /workflowId

- Indexing: Default

- TTL: None

g. ToolLibrary Collection

- Partition key: /workGroupId

- Indexing: Default

- TTL: None

h. Deployments Collection

- Partition key: /workflowId

- Indexing: Default

- TTL: None

i. ExecutionLogs Collection

- Partition key: /deploymentId

- Indexing: Default

- TTL: 180 days

4. Configure the following database settings:

- Throughput: Autoscale with minimum 400 RU/s

- Availability: Zone redundant

- Backup policy: Continuous backup with 7-day retention

- Enable analytical store: Yes

5. Set up Azure AD authentication for the database

6. Create a managed identity for the application to access the database

7. Configure CORS settings to allow access from the application domain

8. Set up Azure Monitor for database metrics and alerts

## Implementation Recommendations

1. **API Layer**: Create an Azure Functions API layer to handle database operations
2. **Authentication**: Use Azure AD B2C for user authentication
3. **Data Migration**: Develop a migration script to convert sample data to the Cosmos DB format
4. **Caching**: Implement Azure Redis Cache for frequently accessed data
5. **Monitoring**: Set up Azure Application Insights for performance monitoring

## Benefits of Database Implementation

1. **Persistence**: Workflows and settings will be saved between sessions
2. **Multi-user Support**: Multiple users can collaborate on workflows
3. **Version Control**: Track changes to workflows over time
4. **Access Control**: Proper implementation of work group permissions
5. **Deployment Tracking**: Monitor workflow deployments and executions
6. **Scalability**: Support for growing number of users and workflows
7. **Analytics**: Ability to analyze workflow usage patterns

Would you like me to elaborate on any specific aspect of this database recommendation or provide more details about implementation strategies?

Feedback submitted

Me

based on the current application what otehr back end systems would you recommend be built for this application

# Recommended Backend Systems for LangGraph Designer

Based on the current application architecture and functionality, I recommend building the following backend systems to support the LangGraph Designer application in Azure:

## 1. Authentication and Authorization Service

**Purpose:** Manage user identity, authentication, and authorization.

**Recommended Azure Services:**

* Azure Active Directory B2C for identity management
* Azure API Management for API security
* Azure Key Vault for secret management

**Key Features:**

* Single sign-on (SSO) integration
* Role-based access control (RBAC)
* Multi-factor authentication
* Token-based authentication with JWT
* Integration with existing enterprise identity systems

## 2. Workflow Execution Engine

**Purpose:** Execute and manage the agent workflows created in the designer.

**Recommended Azure Services:**

* Azure Functions for serverless execution
* Azure Logic Apps for complex workflow orchestration
* Azure Container Instances for isolated execution environments

**Key Features:**

* Real-time workflow execution
* Parallel processing of workflow steps
* Error handling and retry mechanisms
* Execution monitoring and logging
* Support for different agent types and models

## 3. Model Management Service

**Purpose:** Manage LLM models, versioning, and deployment.

**Recommended Azure Services:**

* Azure Machine Learning for model management
* Azure OpenAI Service for model access
* Azure Container Registry for custom model containers

**Key Features:**

* Model registry and versioning
* Model deployment and scaling
* Usage tracking and quotas
* Cost management
* A/B testing capabilities

## 4. Memory and Vector Store Service

**Purpose:** Manage different types of memory for agents and vector storage for retrieval.

**Recommended Azure Services:**

* Azure Cosmos DB for document storage
* Azure Cognitive Search for vector search
* Azure Cache for Redis for short-term memory

**Key Features:**

* Vector embedding storage and retrieval
* Conversation history management
* Knowledge base integration
* Memory type implementations (buffer, summary, entity, etc.)
* Efficient retrieval mechanisms

## 5. Tool Integration Service

**Purpose:** Manage external tool integrations and their execution.

**Recommended Azure Services:**

* Azure API Management for API integration
* Azure Logic Apps for connector management
* Azure Functions for custom tool implementations

**Key Features:**

* Tool registry and discovery
* Secure credential management
* Tool execution and monitoring
* Rate limiting and quota management
* Custom tool development framework

## 6. Deployment and Environment Management

**Purpose:** Manage the deployment of workflows across different environments.

**Recommended Azure Services:**

* Azure DevOps for CI/CD pipelines
* Azure Resource Manager for infrastructure as code
* Azure App Configuration for environment settings

**Key Features:**

* Environment separation (dev, test, prod)
* Deployment approval workflows
* Version control integration
* Rollback capabilities
* Configuration management

## 7. Monitoring and Analytics Service

**Purpose:** Track usage, performance, and provide insights.

**Recommended Azure Services:**

* Azure Application Insights for application monitoring
* Azure Monitor for system monitoring
* Azure Log Analytics for log management
* Power BI for analytics dashboards

**Key Features:**

* Real-time monitoring dashboards
* Performance metrics and alerts
* Usage analytics and trends
* Cost tracking and optimization
* Audit logging

## 8. Work Group Collaboration Service

**Purpose:** Enable collaboration within work groups.

**Recommended Azure Services:**

* Azure Communication Services for real-time collaboration
* Azure SignalR Service for real-time updates
* Azure Storage for shared assets

**Key Features:**

* Real-time collaborative editing
* Comments and feedback system
* Activity feeds and notifications
* Sharing and permissions management
* Version history and comparison

## 9. Content Management System

**Purpose:** Manage documentation, templates, and reusable components.

**Recommended Azure Services:**

* Azure Blob Storage for content storage
* Azure CDN for content delivery
* Azure Static Web Apps for documentation hosting

**Key Features:**

* Documentation management
* Template library
* Component marketplace
* Version control for content
* Search and discovery

## 10. Integration Hub

**Purpose:** Connect with external systems and services.

**Recommended Azure Services:**

* Azure API Management for API gateway
* Azure Event Grid for event-driven architecture
* Azure Service Bus for message queuing

**Key Features:**

* Webhook management
* API gateway for external services
* Event subscription and publishing
* Message queue for asynchronous processing
* Integration with enterprise systems

## Implementation Strategy

I recommend a phased approach to implementing these backend systems:

1. **Phase 1:** Core Infrastructure
   * Authentication Service
   * Database Implementation
   * Basic API Layer
2. **Phase 2:** Workflow Capabilities
   * Workflow Execution Engine
   * Model Management Service
   * Memory Service
3. **Phase 3:** Collaboration Features
   * Work Group Collaboration Service
   * Deployment Management
   * Monitoring and Analytics
4. **Phase 4:** Advanced Features
   * Tool Integration Service
   * Content Management System
   * Integration Hub

This approach allows for incremental development and deployment while providing immediate value to users at each phase.

Would you like me to elaborate on any specific backend system or discuss implementation details for a particular component?

Feedback submitted

Me

yes please provide more detail on all of them

# Detailed Backend Systems for LangGraph Designer

## 1. Authentication and Authorization Service

### Architecture Components

* **Identity Provider**: Azure AD B2C for managing user identities, supporting social logins, and enterprise federation
* **Authentication API**: RESTful API for login, logout, token refresh, and password reset
* **Authorization Service**: Microservice for permission management and access control
* **User Profile Service**: Manages user preferences, settings, and profile information

### Technical Implementation

* **JWT Implementation**: Issue short-lived access tokens (1 hour) and longer refresh tokens (14 days)
* **Permission Model**:

json

CopyInsert

{

"userId": "string",

"workGroupPermissions": [

{

"workGroupId": "string",

"role": "Admin|Editor|Viewer",

"entityPermissions": [

{

"entityType": "Skill|Team|Agent|Workflow",

"entityId": "string",

"permissions": ["Read", "Write", "Execute", "Share"]

}

]

}

],

"globalPermissions": ["CreateWorkGroup", "ManageUsers"]

}

* **Integration Points**:
  + Azure AD B2C for identity management
  + Azure API Management for securing APIs
  + Azure Key Vault for storing secrets and certificates

### Development Considerations

* Implement PKCE flow for SPA authentication
* Support for enterprise SSO via SAML and OpenID Connect
* Implement rate limiting to prevent brute force attacks
* Provide audit logging for all authentication events

## 2. Workflow Execution Engine

### Architecture Components

* **Workflow Parser**: Converts visual workflow to executable code
* **Execution Coordinator**: Manages the execution of workflow steps
* **State Manager**: Maintains workflow state during execution
* **Agent Runtime**: Executes agent nodes with appropriate LLM models
* **Tool Executor**: Safely executes tool code with proper isolation

### Technical Implementation

* **Workflow Representation**:

json

CopyInsert

{

"nodes": [

{

"id": "node1",

"type": "agent",

"config": { "model": "gpt-4o", "temperature": 0.7 }

}

],

"edges": [

{

"source": "node1",

"target": "node2",

"condition": "success"

}

],

"entryPoints": ["node1"],

"exitPoints": ["node5"]

}

* **State Management**:
  + Distributed state using Azure Cosmos DB
  + Checkpointing for long-running workflows
  + Transaction support for atomic operations
* **Execution Models**:
  + Synchronous execution for simple workflows
  + Asynchronous execution with webhooks for long-running workflows
  + Scheduled execution using Azure Functions timer triggers

### Development Considerations

* Implement circuit breakers for external service calls
* Add observability through distributed tracing
* Support for execution timeouts and cancellation
* Implement cost tracking and budget enforcement

## 3. Model Management Service

### Architecture Components

* **Model Registry**: Catalog of available models with metadata
* **Model Proxy**: Unified API for different model providers
* **Quota Manager**: Enforces usage limits and tracks consumption
* **Model Cache**: Caches model responses for efficiency
* **Prompt Template Manager**: Stores and versions prompt templates

### Technical Implementation

* **Model Registry Schema**:

json

CopyInsert

{

"id": "string",

"name": "string",

"provider": "OpenAI|Anthropic|Azure|Custom",

"capabilities": ["chat", "embedding", "function-calling"],

"contextWindow": 16000,

"tokenCostInput": 0.0001,

"tokenCostOutput": 0.0003,

"maxTokens": 4000,

"deprecated": false,

"availableRegions": ["eastus", "westeurope"]

}

* **Prompt Template System**:
  + Support for variables and conditional logic
  + Version control for templates
  + A/B testing capabilities
* **Integration Points**:
  + Azure OpenAI Service for model hosting
  + Azure Machine Learning for custom model deployment
  + Azure Container Registry for model containers

### Development Considerations

* Implement fallback mechanisms for model unavailability
* Add support for model fine-tuning and customization
* Create a model evaluation framework
* Implement content filtering and safety measures

## 4. Memory and Vector Store Service

### Architecture Components

* **Memory Manager**: Coordinates different memory types
* **Vector Database**: Stores and retrieves vector embeddings
* **Embedding Service**: Converts text to vector embeddings
* **Memory Type Implementations**: Different memory strategies
* **Retrieval Engine**: Optimized retrieval algorithms

### Technical Implementation

* **Memory Types**:
  + Conversation Buffer: Recent messages stored in Redis
  + Summary Memory: Periodic summaries stored in Cosmos DB
  + Entity Memory: Structured entity data in Cosmos DB
  + Vector Memory: Embeddings in Azure Cognitive Search
* **Vector Schema**:

json

CopyInsert

{

"id": "string",

"text": "string",

"embedding": [0.1, 0.2, ...],

"metadata": {

"source": "string",

"timestamp": "datetime",

"workflowId": "string"

}

}

* **Retrieval Methods**:
  + Semantic search with configurable relevance thresholds
  + Hybrid search combining keyword and semantic approaches
  + Time-weighted retrieval for recency bias

### Development Considerations

* Implement TTL policies for different memory types
* Add support for memory compression techniques
* Create memory visualization tools
* Implement privacy controls and data retention policies

## 5. Tool Integration Service

### Architecture Components

* **Tool Registry**: Catalog of available tools
* **Tool Executor**: Safely runs tool code
* **Credential Manager**: Securely stores and provides tool credentials
* **Tool Builder**: Interface for creating custom tools
* **Tool Marketplace**: Sharing and discovery of tools

### Technical Implementation

* **Tool Definition Schema**:

json

CopyInsert

{

"id": "string",

"name": "string",

"description": "string",

"version": "string",

"inputSchema": {

"type": "object",

"properties": {}

},

"outputSchema": {

"type": "object",

"properties": {}

},

"implementation": {

"type": "azure\_function|container|openapi",

"location": "string",

"authType": "none|api\_key|oauth"

},

"rateLimit": {

"requestsPerMinute": 60

}

}

* **Execution Environment**:
  + Sandboxed execution using Azure Container Instances
  + Timeout and resource limits
  + Input validation against schema
* **Integration Methods**:
  + Direct function calls for Azure Functions
  + REST API calls for web services
  + Container execution for custom tools

### Development Considerations

* Implement tool versioning and compatibility checking
* Add support for tool composition and chaining
* Create a testing framework for tools
* Implement usage analytics and popularity metrics

## 6. Deployment and Environment Management

### Architecture Components

* **Environment Manager**: Manages different deployment environments
* **Deployment Pipeline**: Automates the deployment process
* **Configuration Service**: Manages environment-specific settings
* **Release Manager**: Coordinates version releases
* **Rollback Service**: Handles deployment failures

### Technical Implementation

* **Environment Definition**:

json

CopyInsert

{

"id": "string",

"name": "dev|staging|production",

"resources": {

"database": {

"connectionString": "secret:cosmos-connection"

},

"storage": {

"connectionString": "secret:storage-connection"

}

},

"features": {

"enableExperimentalTools": true,

"maxConcurrentExecutions": 10

}

}

* **Deployment Process**:
  + Validation of workflow integrity
  + Automated testing of critical paths
  + Blue/green deployment for zero downtime
  + Canary releases for gradual rollout
* **Configuration Management**:
  + Environment-specific variables
  + Feature flags for controlled rollout
  + Secrets management via Key Vault

### Development Considerations

* Implement approval workflows for production deployments
* Add deployment metrics and health checks
* Create deployment notifications and alerts
* Implement infrastructure as code for environment consistency

## 7. Monitoring and Analytics Service

### Architecture Components

* **Telemetry Collector**: Gathers performance and usage data
* **Log Aggregator**: Centralizes logs from all services
* **Metrics Dashboard**: Visualizes key performance indicators
* **Alerting System**: Notifies of issues and anomalies
* **Analytics Engine**: Provides insights and trends

### Technical Implementation

* **Telemetry Schema**:

json

CopyInsert

{

"timestamp": "datetime",

"userId": "string",

"workflowId": "string",

"eventType": "execution\_start|node\_execution|error|completion",

"duration": 1200,

"resourceUsage": {

"tokens": 1500,

"computeSeconds": 2.5

},

"metadata": {}

}

* **Monitoring Dashboards**:
  + Real-time execution monitoring
  + Resource usage and cost tracking
  + Error rates and performance metrics
  + User activity and adoption metrics
* **Alerting Rules**:
  + Error rate thresholds
  + Performance degradation
  + Quota approaching limits
  + Unusual activity patterns

### Development Considerations

* Implement distributed tracing for request flows
* Add custom metric collection for business KPIs
* Create exportable reports for stakeholders
* Implement predictive analytics for resource planning

## 8. Work Group Collaboration Service

### Architecture Components

* **Collaboration Hub**: Central service for real-time collaboration
* **Notification Service**: Manages user notifications
* **Activity Feed**: Tracks and displays user actions
* **Comment System**: Allows discussions on workflows
* **Sharing Service**: Manages resource sharing

### Technical Implementation

* **Real-time Updates**:
  + WebSocket connections via Azure SignalR
  + Event-based notifications
  + Presence awareness for active users
* **Activity Schema**:

json

CopyInsert

{

"id": "string",

"timestamp": "datetime",

"userId": "string",

"userName": "string",

"actionType": "create|edit|delete|deploy|share",

"resourceType": "workflow|agent|tool",

"resourceId": "string",

"resourceName": "string",

"details": {}

}

* **Notification System**:
  + In-app notifications
  + Email digests
  + Custom notification preferences
  + Read/unread status tracking

### Development Considerations

* Implement conflict resolution for simultaneous edits
* Add support for @mentions and assignments
* Create notification batching to prevent overload
* Implement read receipts for critical notifications

## 9. Content Management System

### Architecture Components

* **Document Store**: Manages documentation and content
* **Template Library**: Stores reusable workflow templates
* **Asset Manager**: Handles media and resource files
* **Search Engine**: Enables content discovery
* **Version Control**: Tracks content changes

### Technical Implementation

* **Content Schema**:

json

CopyInsert

{

"id": "string",

"type": "document|template|asset",

"title": "string",

"content": "string",

"format": "markdown|html|json",

"tags": ["string"],

"createdBy": "string",

"createdAt": "datetime",

"updatedAt": "datetime",

"version": "string",

"status": "draft|published|archived"

}

* **Template System**:
  + Parameterized workflow templates
  + Category-based organization
  + Rating and popularity metrics
  + Import/export capabilities
* **Search Capabilities**:
  + Full-text search with relevance ranking
  + Faceted search by content type, tags, author
  + Recent and popular content suggestions

### Development Considerations

* Implement content approval workflows
* Add support for rich media embedding
* Create content analytics for usage tracking
* Implement content localization support

## 10. Integration Hub

### Architecture Components

* **API Gateway**: Entry point for external integrations
* **Webhook Manager**: Handles incoming and outgoing webhooks
* **Event Bus**: Manages event publishing and subscription
* **Connector Library**: Pre-built integrations with common services
* **Custom Integration Builder**: Tools for creating new integrations

### Technical Implementation

* **Webhook Configuration**:

json

CopyInsert

{

"id": "string",

"name": "string",

"url": "string",

"events": ["workflow.started", "workflow.completed"],

"headers": {

"Authorization": "secret:webhook-auth"

},

"retryPolicy": {

"maxAttempts": 3,

"backoffSeconds": 5

},

"active": true

}

* **Event System**:
  + Publish-subscribe pattern using Azure Event Grid
  + Event filtering and routing
  + Dead-letter queue for failed deliveries
* **Integration Types**:
  + REST API integrations
  + Message queue integrations
  + Database connectors
  + Custom protocol adapters

### Development Considerations

* Implement rate limiting for API endpoints
* Add circuit breakers for external service calls
* Create integration monitoring and health checks
* Implement versioning for API endpoints

## Implementation Roadmap

### Phase 1: Foundation (Months 1-3)

1. Set up Azure Cosmos DB
2. Implement Authentication Service
3. Create basic API layer
4. Develop core workflow storage

### Phase 2: Core Functionality (Months 4-6)

1. Build Workflow Execution Engine
2. Implement Model Management Service
3. Develop Memory Service
4. Create basic monitoring

### Phase 3: Collaboration (Months 7-9)

1. Implement Work Group Collaboration
2. Build Deployment Management
3. Enhance Monitoring and Analytics
4. Develop User Management

### Phase 4: Advanced Features (Months 10-12)

1. Build Tool Integration Service
2. Implement Content Management
3. Develop Integration Hub
4. Create Marketplace functionality

## Technology Stack Recommendation

* **Backend APIs**: .NET Core or Node.js with TypeScript
* **Serverless Functions**: Azure Functions
* **Workflow Orchestration**: Azure Durable Functions
* **Databases**: Azure Cosmos DB, Azure SQL Database
* **Search**: Azure Cognitive Search
* **Caching**: Azure Redis Cache
* **Messaging**: Azure Service Bus, Azure Event Grid
* **Authentication**: Azure AD B2C
* **DevOps**: Azure DevOps, GitHub Actions
* **Monitoring**: Application Insights, Azure Monitor

This comprehensive backend system will transform the LangGraph Designer from a standalone application into a robust, enterprise-ready platform for building and deploying AI agent workflows.

Feedback submitted

Generating.

9 minutes ago

5 minutes ago

Start with History Ctrl + ⏎

Press Enter again to interrupt and send a new message

Start with History Ctrl + ⏎

Deploy your application

Done