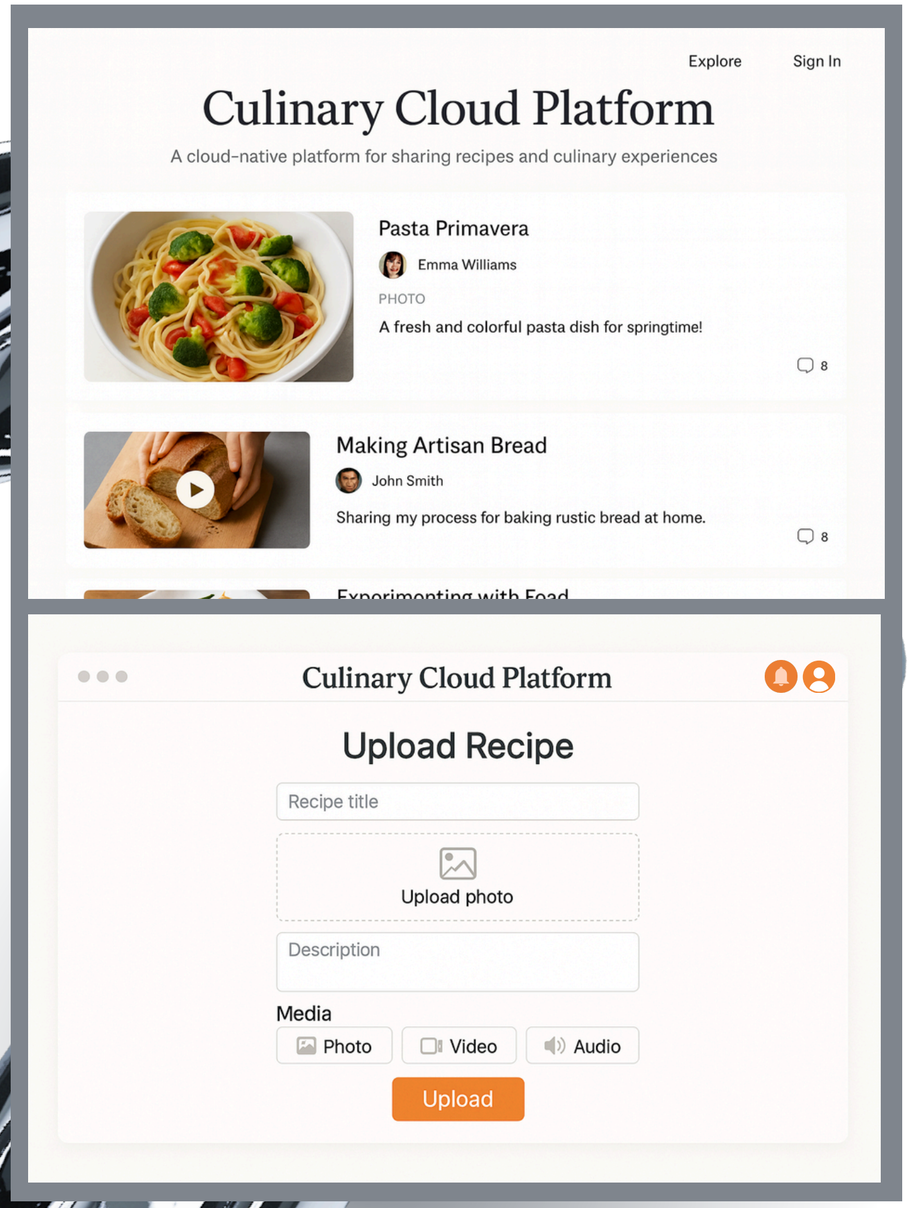
# CULINARY CLOUD PLATFORM

Share recipes. Tell stories. Inspire flavour - All powered by the Cloud.

**Student Name -** Indreswar Balaji
**Student ID -** B00959342




# PROBLEM DEFINITION

Modern food lovers share their recipes and cooking experiences across multiple platforms, from Instagram photos to YouTube cooking videos. But these are scattered, inconsistent, and lack a unified experience.

There is no dedicated space where users can combine recipes, photos, videos, and voice notes within one accessible and scalable platform.

## Challenges Identified

**Multimedia storage issues: Handling large volumes of images, videos, and audio efficiently.**
**Scalability: Traditional web hosting cannot dynamically handle high traffic during peak activity (e.g., viral recipes).**
**Data management: Storing and querying user posts, comments, and tags requires a distributed, flexible database.**
**Global accessibility: Users from different regions expect fast content delivery with minimal latency.**

To design a cloud-native platform for sharing and engaging with multimedia recipes, ensuring scalability, efficiency, and global reach using **Azure services**.

# AIM & OBJECTIVES

**Aim:** To design a cloud-native platform for sharing and engaging with multimedia recipes, ensuring scalability, efficiency, and global reach using **Azure services.**

## Objectives:

1. Design a scalable cloud architecture using **Azure Functions**, Blob Storage, and Cosmos DB.
2. Enable multimedia uploads such as photos, videos, and voice recordings.
3. Develop REST APIs for posts, comments, and media handling.
4. Implement an event-driven workflow for background tasks like thumbnail or transcript generation.
5. Host a responsive web interface through **Azure Static Web Apps** for global accessibility.
6. Ensure monitoring and optimization using **Azure Application Insights** and cost-control strategies.

# CLOUD-NATIVE JUSTIFICATION

Building the Culinary Cloud Platform as a cloud-native application ensures scalability, high availability, and cost efficiency while minimizing operational overhead.

**Elastic Scalability:** Azure Functions automatically scale up during peak recipe uploads and scale down when traffic is low - saving costs.

**Efficient Multimedia Storage:** Azure Blob Storage provides durable, pay-as-you-go storage for photos, videos, and audio files.

**Event-Driven Architecture:** Azure Event Grid triggers background automation like generating thumbnails or voice-to-text transcripts.

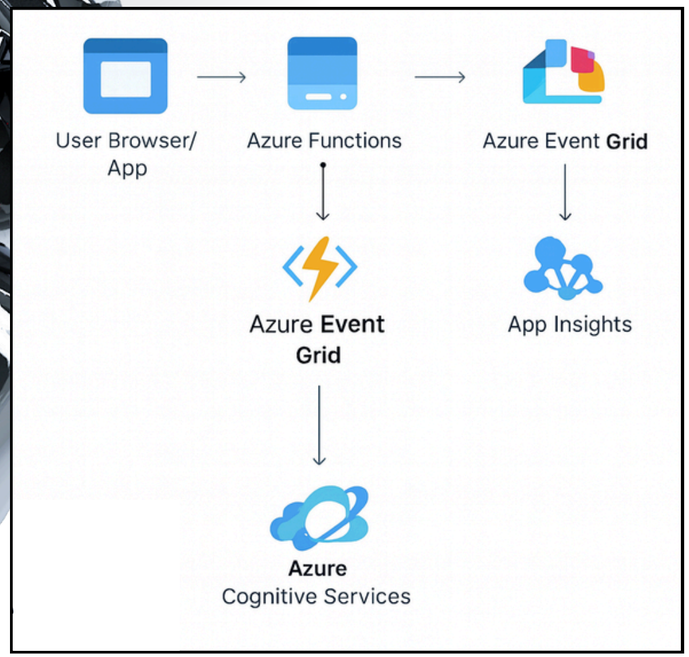
**Serverless Deployment:** No need for manual server management — Functions, Static Web Apps, and Logic Apps handle infrastructure automatically.

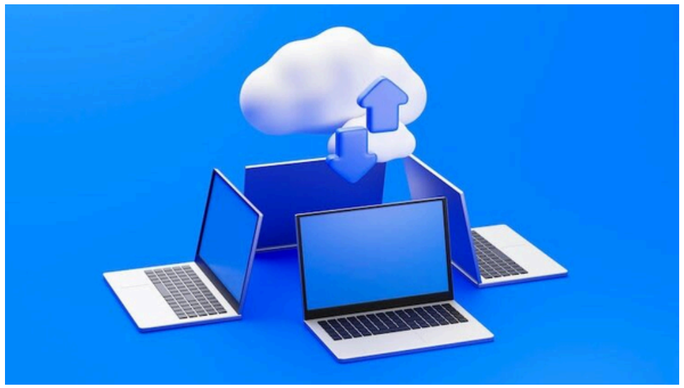
**Global Availability:** Azure CDN and Cosmos DB’s geo-replication ensure fast, reliable content delivery worldwide.

**Cost Efficiency:** Pay only for compute and storage used — ideal for a community platform with variable usage patterns.

# HIGH-LEVEL ARCHITECTURE DESIGN

The function of each Azure component used in Culinary Cloud Platform and how they integrate together.

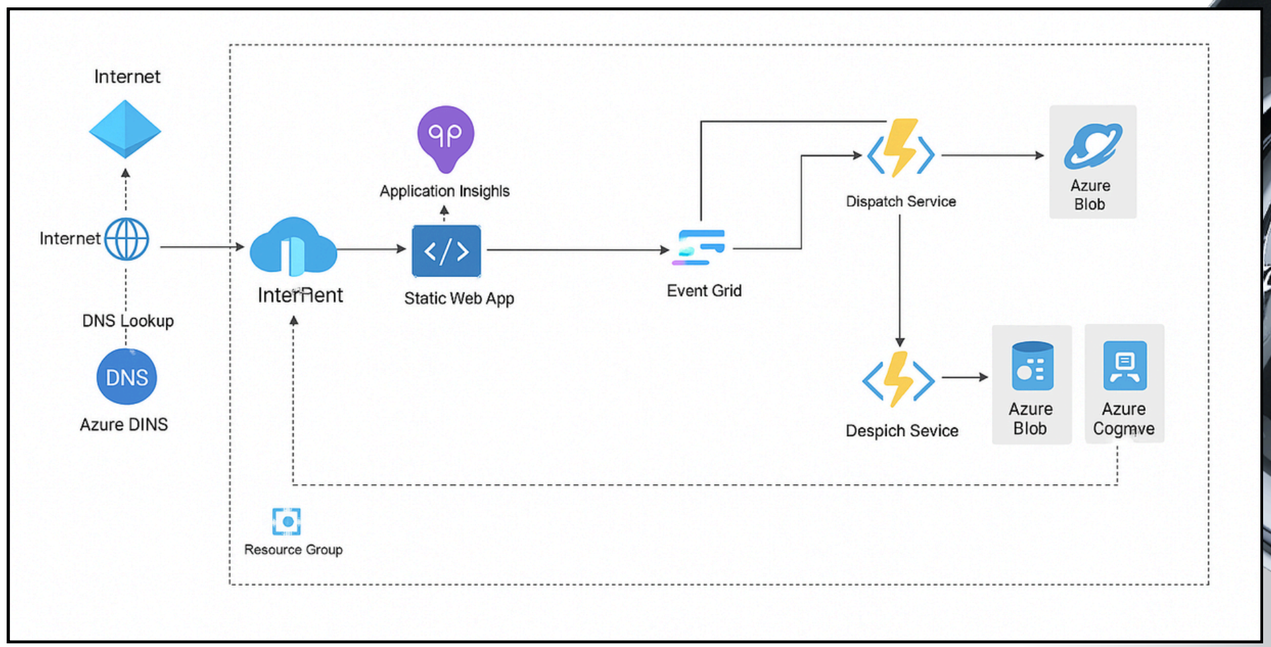


**User Interaction** - User Browser / App
**API Handling** - User request like posts, videos, audio recordings.
**Automation (Event-Driven)** - New uploads trigger Azure Event Grid

- Invoking Functions
- Generating thumbnail
- Running AI tagging

**Global Distribution** - Azure CDN ensures fast delivery

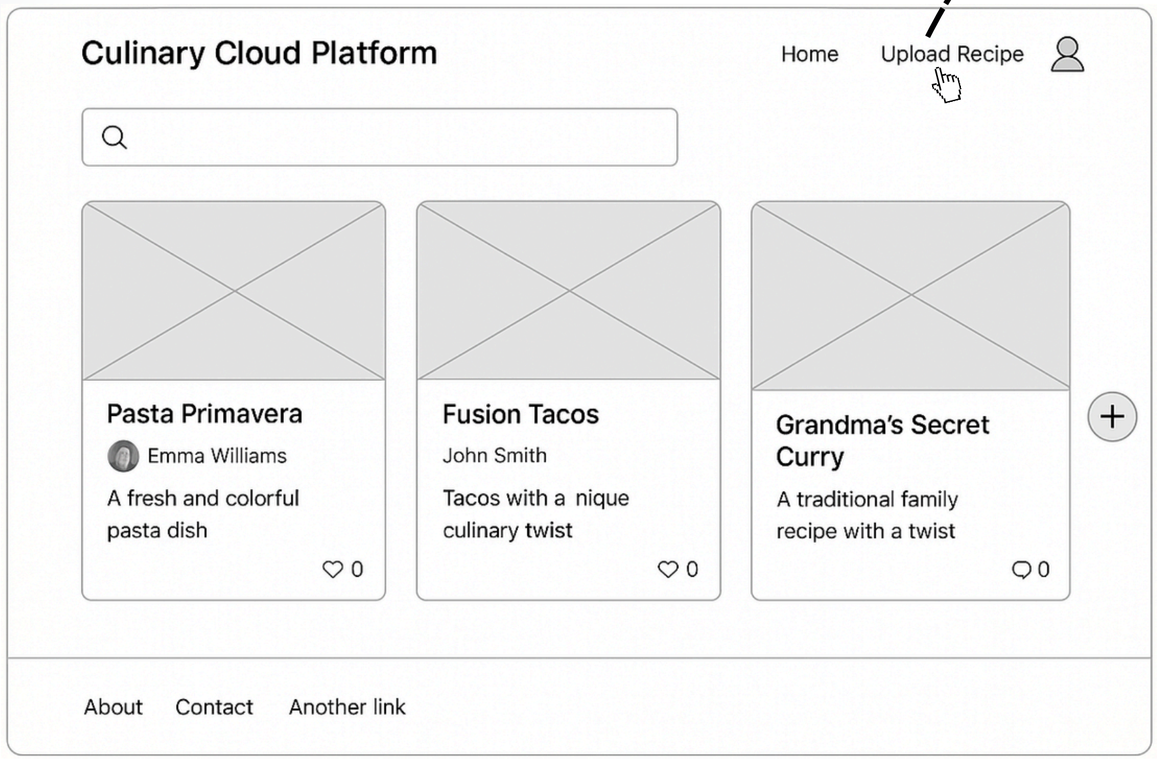


# DETAILED COMPONENT VIEW

| | |
|---|---|
| Azure Functions | Hosts the platform’s web interface where users can browse recipes, upload photos, videos, and voice notes. Provides global access and integrates directly with backend APIs. |
| Azure Functions | Handles all core logic such as recipe submission, user requests, data validation, and retrieval. Automatically scales to handle peak traffic without server management. |
| Azure Blob Storage | Stores user-uploaded multimedia files — images, videos, and voice notes. Triggers events through Event Grid when a new file is uploaded. |
| Azure Cosmos DB (NoSQL) | Stores structured recipe data, user profiles, comments, and metadata. Ensures low-latency, globally distributed access. |
| Azure Event Grid + Logic Apps | Manages background workflows — triggers Azure Functions to generate thumbnails, perform AI tagging, or send notifications when new content is uploaded. |
| Azure Cognitive Services | Uses Computer Vision or Speech API for automatic image ingredient tagging or converting voice recipes to text. Enhances accessibility and discoverability. |
| Azure Application Insights | Collects logs, tracks performance metrics, monitors backend function performance, and identifies bottlenecks or failed uploads. |
| Azure CDN (via Static Web App) | Ensures fast delivery of web content to users around the world, improving the user experience. |

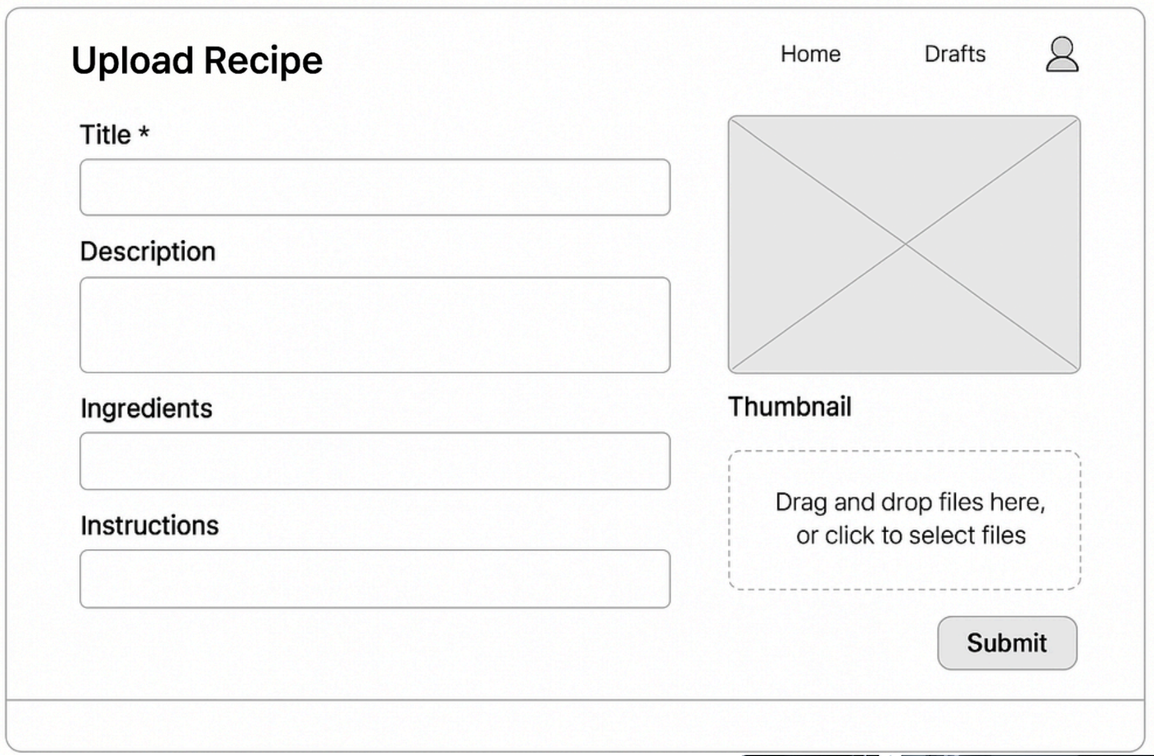
# WIREFRAME AND UI DESIGN

## MAIN PAGE




The main page lets users browse, search, and share recipes easily with a clean, user-friendly layout.

## UPLOAD PAGE




Users can create and manage recipes by adding titles, details, and media, with options to save drafts before publishing.

# DATA MODEL AND SCHEMA DESIGN

**Key Entities**

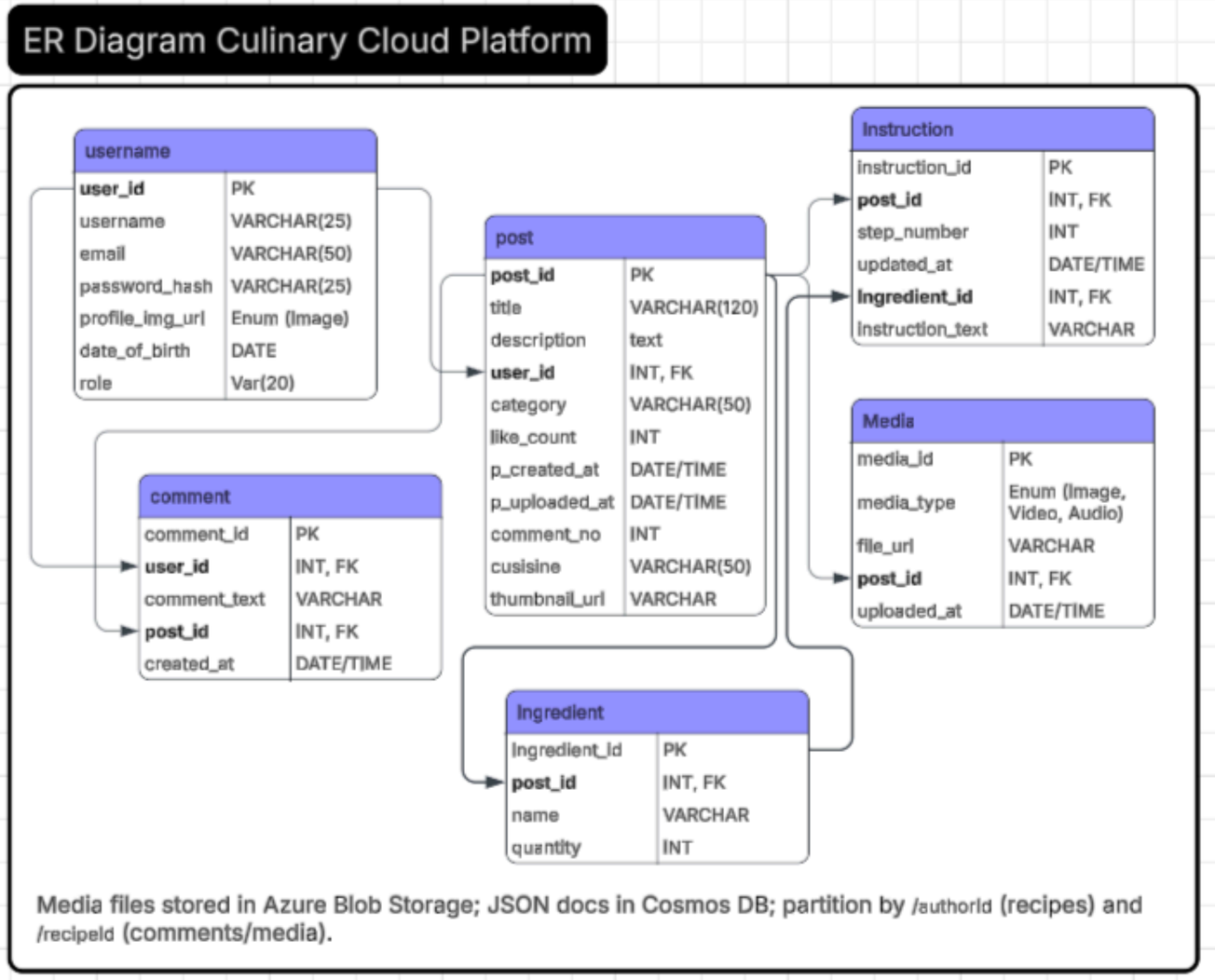
- **User** – Stores user credentials, profile image, role, and personal details.
- **Post** (Recipe) – Main table containing title, description, category, and metadata such as likes and comments count.
- **Ingredient** – Lists ingredients for each recipe with quantity and unit details.
- **Instruction** – Defines step-by-step preparation with optional linked media.
- **Media** – Stores URLs and metadata of uploaded files (image, video, or audio).
- **Comment** – Allows users to provide feedback on recipes, linked to both post and user.

**Relationships**

- **User** → **Post**: One-to-Many (each user can publish multiple recipes).
- **Post** → **Ingredient** / **Instruction** / **Media** / **Comment**: One-to-Many relationships.
- **User** → **Comment**: One-to-Many (users can comment on multiple recipes).
- All connections use foreign keys (FK) to ensure referential integrity.

**Storage Approach**

- **Azure Cosmos DB** stores the structured recipe, comment, and user data as JSON documents.
- **Azure Blob Storage** holds multimedia content like photos, videos, and audio linked by file URLs.
- Each post references its media and related entities through unique IDs for efficient retrieval.



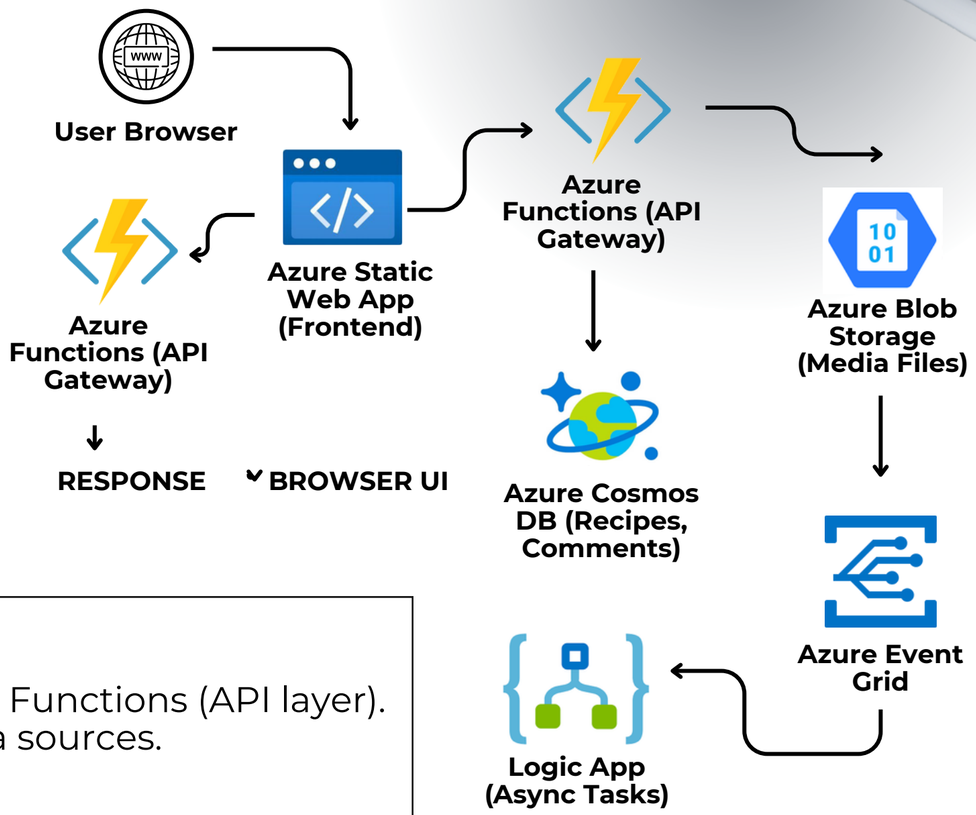

This structure ensures efficient data storage, easy retrieval, and scalability across **Azure services**.

# REST API AND DATA FLOW DESIGN

The **REST API** is implemented using **Azure Functions** with **HTTP triggers**, enabling a serverless backend that handles **CRUD operations** for recipes, comments, and media files. Each request is authenticated, validated, and routed to the appropriate data store (Cosmos DB or Blob Storage).

| Operation | Method | Endpoint | Azure Service |
|---|---|---|---|
| **Create Recipe** | **POST** | /api/recipes | Azure Function → Cosmos DB |
| **Get Recipe** | **GET** | /api/recipes/{id} | Azure Function → Cosmos DB |
| **Upload Media** | **POST** | /api/media/upload-url | Azure Function → Blob Storage |
| **Add Comment** | **POST** | /api/recipes/{id}/comments | Azure Function → Cosmos DB |
| **Like Recipe** | **POST** | /api/recipes/{id}/like | Azure Function → Cosmos DB |
| **Trigger Automation** | **POST** | /events/blob-created | Event Grid → Logic App |

Serverless REST API design enabling secure CRUD operations via Azure Functions and seamless data flow between frontend, storage, and database.




**Data Flow Explanation**

1. **User Action**: User uploads or views a recipe from the web app.
2. **Frontend (Azure Static Web App)**: Sends an HTTP request to Azure Functions (API layer).
3. **Azure Functions**: Processes request → validates → interacts with data sources.
   - For media uploads → stores file in **Blob Storage**.
   - For recipe or comment data → writes/reads from **Cosmos DB**.
4. **Azure Event Grid (Async Trigger)**: On new upload, triggers background automation (thumbnail or AI tagging).
5. **Response**: Function returns a JSON response to the frontend for rendering.

# EVENT-DRIVEN WORKFLOW AND AUTOMATION

The **Culinary Cloud Platform** uses an event-driven architecture to handle asynchronous background processes such as: Generating image thumbnails, Extracting audio transcripts, Tagging recipes using AI, Sending notifications or updates to users.
This reduces manual workload and improves system responsiveness.

**1.User Uploads Media:**

The user uploads a recipe photo, video, or voice note.
Azure Functions stores the file in Azure Blob Storage.

**2.Event Trigger (Blob Created):**

The Blob Storage event (BlobCreated) automatically triggers Azure Event Grid.

**3.Automation Execution:**

Azure Logic App (or another Azure Function) runs background tasks like:

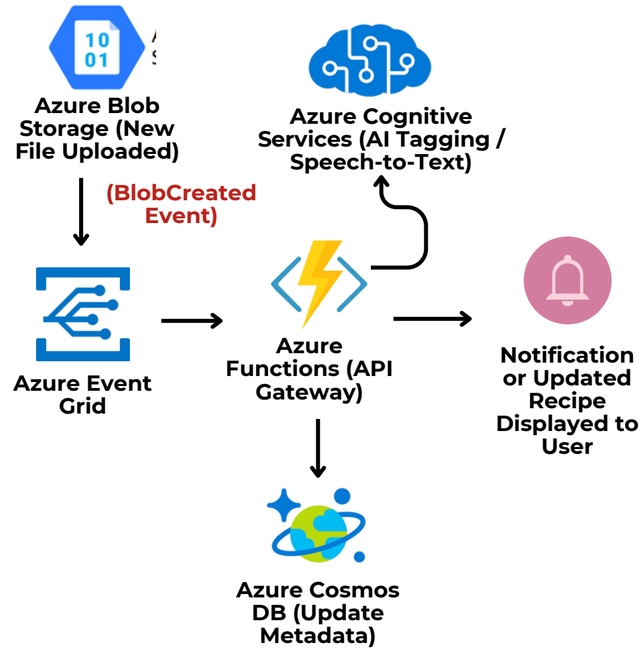
- Generate a thumbnail
- Call Azure Cognitive Services for AI tagging or speech-to-text
- Send a notification to the user

**4.Data Update:**

Once automation completes, Logic App updates Azure Cosmos DB (adds thumbnail URL/ AI tags).

**5.User Feedback**

Frontend fetches the updated data via REST API to show the enhanced recipe card.



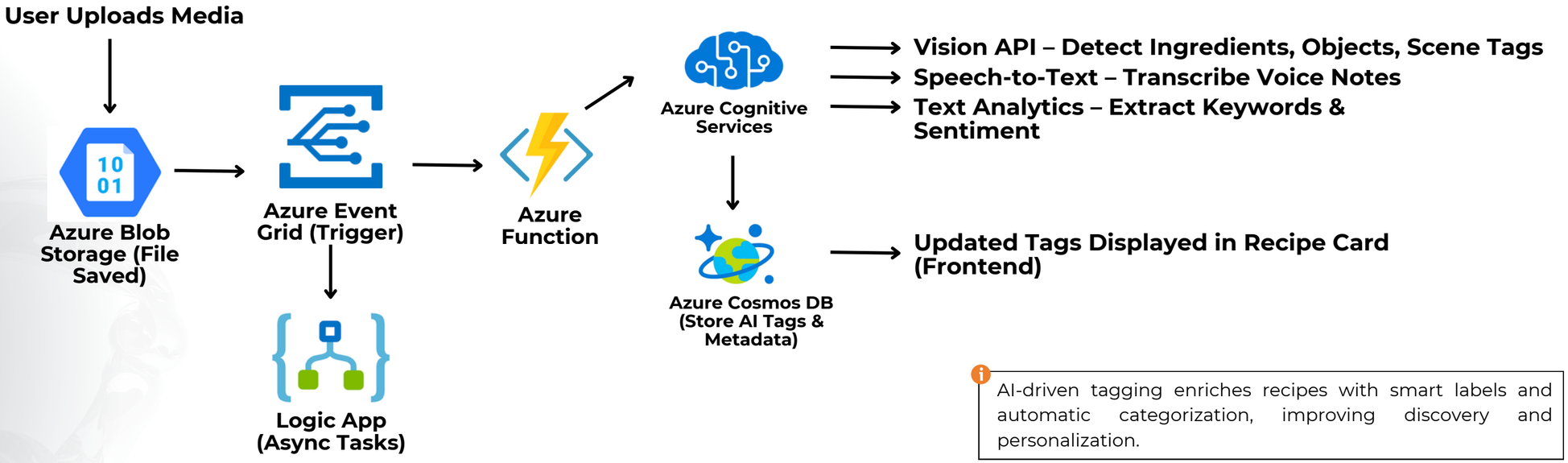

Event-driven workflow automates background tasks like media processing, AI tagging, and data updates using Azure Event Grid and Logic Apps.

# ADVANCED FEATURE:
# AI-POWERED RECIPE TAGGING

The **Culinary Cloud Platform** integrates Azure Cognitive Services to automate the process of analyzing uploaded media and tagging recipes intelligently.

When a user uploads a recipe photo, video, or voice note:

- The system automatically **detects** relevant objects, ingredients, or keywords.
- These **tags** are stored in **Azure Cosmos DB** to improve search, filtering, and personalization.
- Speech-to-text transcribes voice notes into instructions.
- Image analysis generates descriptive tags and categories.



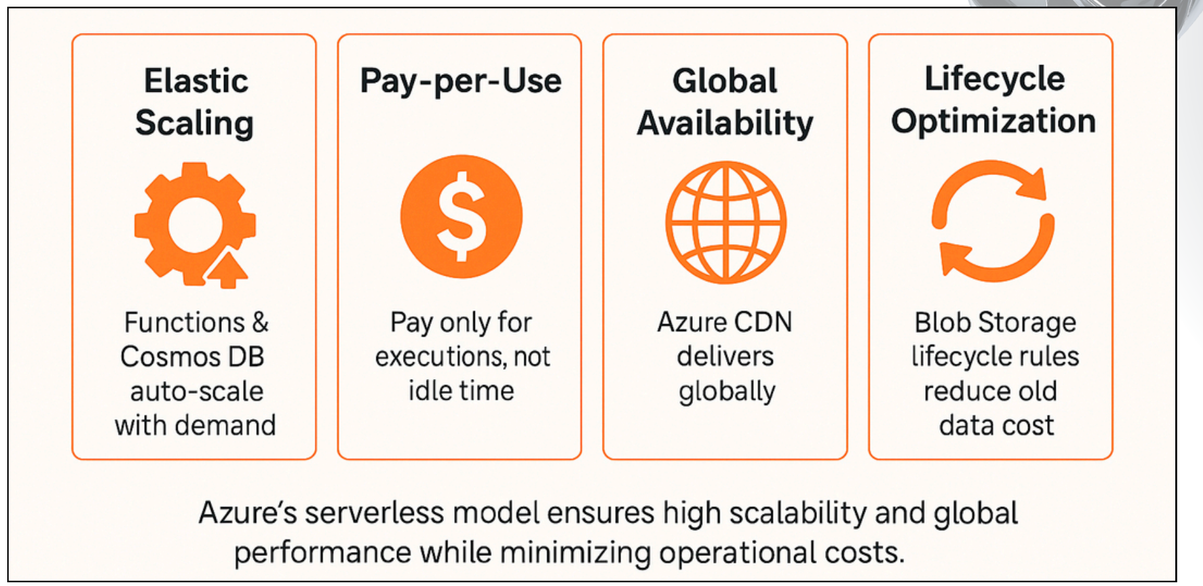

AI-driven tagging enriches recipes with smart labels and automatic categorization, improving discovery and personalization.

# SCALABILITY AND COST OPTIMIZATION

The **Culinary Cloud Platform** automatically scales resources based on user activity and workload demand. By using Azure's serverless services, the system ensures cost-effective scaling, paying only for active usage while maintaining global availability.

Azure's serverless and distributed design ensures high scalability and cost-efficiency.

- Azure Functions and Cosmos DB automatically scale with demand.
- Pay-per-use billing minimizes idle costs.
- CDN and Blob Storage lifecycle management reduce global latency and storage overhead.



# RELIABILITY, SECURITY, AND MONITORING

The platform uses Azure's integrated services to ensure data protection, continuous availability, and proactive monitoring. By combining redundancy, role-based access, and observability, the system remains secure, resilient, and compliant with data regulations.

**Reliability**

- Built on Azure's global infrastructure with geo-redundant storage.
- Azure Front Door and CDN ensure low-latency delivery worldwide.
- Cosmos DB multi-region replication provides high availability and fault tolerance.
- Serverless Functions automatically restart on failure and scale seamlessly.

**Monitoring & Observability**

- Azure Application Insights monitors latency, errors, and API performance.
- Log Analytics and Dashboard Alerts provide real-time insights.
- Failure alerts and Function telemetry help detect and resolve issues proactively.

**Security**

- Authentication: User logins protected via hashed passwords or Azure Entra ID (optional).
- Authorization: Role-based access (User, Admin).
- Data Protection:
  - TLS 1.3 for all data in transit.
  - AES-256 encryption for data at rest (Blob + Cosmos DB).
- Blob SAS Tokens for secure temporary media access.
- GDPR compliance – users can request data deletion (soft delete mechanism).

# LIMITATIONS AND FUTURE ENHANCEMENTS

Every cloud-native system evolves through iteration. While the current Culinary Cloud Platform establishes a scalable, serverless foundation, several opportunities remain to refine performance, intelligence, and user experience.

### Current Limitations

Despite its robust Azure backbone, the platform lacks real-time collaboration and offline support, which limits user engagement on low-connectivity devices.
AI-powered media analysis relies on external Cognitive Services, leading to potential cost spikes with high-volume uploads.
Security and identity management are functional but could be expanded with deeper Azure Entra ID integration and multi-factor authentication for enterprise-grade adoption.

### Planned Enhancements

The next development phase focuses on personalization and community features.
Upcoming additions include AI-driven recipe recommendations, real-time chat and comment threads, and a multi-language translation layer to broaden accessibility.
Performance improvements will be achieved through edge caching, content pre-fetching, and adaptive scaling policies that adjust compute resources dynamically.

### Current Limitations

Long-term growth aims to transform the platform into a sustainable, intelligent culinary ecosystem.
By introducing microservices orchestration via Kubernetes, implementing tiered storage automation, and leveraging AI-based anomaly detection, the system will remain resilient, cost-efficient, and future-ready.

# CONCLUSION & REFERENCES

## Conclusion

The Culinary Cloud Platform demonstrates how a cloud-native, serverless architecture can deliver an interactive and scalable environment for recipe sharing.
By integrating Azure Static Web Apps, Functions, Cosmos DB, and Blob Storage, the platform efficiently manages multimedia content while ensuring scalability, security, and global availability.
The use of Azure Cognitive Services introduces intelligence through automated recipe tagging and transcription, enhancing both usability and personalization.
This foundation prepares the system for Coursework 2 (CW2), where the architecture will evolve into a fully functional prototype hosted on Azure.

## References


1. Microsoft (2024) Azure Architecture Center. Available at: https://learn.microsoft.com/en-us/azure/architecture (Accessed: 31 October 2025).
2. Microsoft (2024) Azure Cosmos DB documentation. Available at: https://learn.microsoft.com/en-us/azure/cosmos-db (Accessed: 31 October 2025).
3. Microsoft (2024) Azure Event Grid documentation. Available at: https://learn.microsoft.com/en-us/azure/event-grid (Accessed: 31 October 2025).
4. Microsoft (2024) Azure Cognitive Services overview. Available at: https://learn.microsoft.com/en-us/azure/cognitive-services (Accessed: 31 October 2025).
5. Microsoft (2024) Designing Cloud-Native Applications on Azure. Available at: https://learn.microsoft.com/en-us/azure (Accessed: 31 October 2025).
6. Ulster University (2025) COM682 Cloud-Native Development Module Resources. London: Ulster University.

## Appendix III. Coursework 1 Submission Checklist

*Before submitting, make sure you have addressed the following:*

***General***

- *Did you include a **title slide** with project name, one-line description, your name, and student number?*
- *Is your submission in **PowerPoint format** and within **15 slides** (excluding references)?*

***Problem & Context***

- *Did you **discuss the problem** you are solving?*
- *Did you **identify issues related to scalability** of resources in your project?*

***Solution Design***

- *Did you include a **solution architecture diagram** using only **Azure resources**?*
- *Did you design and present a **multimedia hosting webpage layout** (wireframe or design diagram)?*
- *Did you include a **network/architecture diagram** showing compute, storage, networking, etc.?*
- *Did you design a **database schema** (list of resources and/or ERD diagram) for storing user entries?*
- *Did you design the **REST API** for CRUD operations and present it in a **UML diagram**?*
- *Did you integrate (or at least propose integration of):*
    - *Static HTML hosting of a webpage frontend?*
    - *REST endpoints for service logic and storage connections.*
    - *Use of both **SQL** and **NoSQL** database structures?*

***Critical Evaluation***

- *Did you provide an **overview of advanced features** you intend to develop in the final solution?*
- *Did you include an **assessment of limitations** of your designed solution?*
- *Did you include an **assessment of scalability** (horizontal/vertical scaling, autoscaling, etc.)?*
- *Did you justify **why you selected specific Azure services** over alternatives?*

***Conclusion & References***

- *Did you include **concluding comments** summarising your design?*
- *Did you provide **references** to sources, frameworks, or Azure documentation you used?*

☑ *If you can tick all these boxes, your submission is likely to meet the expectations.*

*Please submit this checklist at the end of the PowerPoint slide deck. Checklist slide will not be counted in the 15 slides limit.*