Technical Documentation: Data Management

1. Updated Database Schema Design **3**
Ingredient Reference Table **3**
Recipes Table **3**
RecipeIngredients Table (Junction Table) **3**
Instructions Table **3**
Images Table **4**
2. Updated Approach for Ingredient Management **4**
2.1 Ingredient Standardization **4**
2.2 Fuzzy Matching with User Prompt 4
3. API Integration for Nutritional Analysis **4**
4. Data Synchronization Strategy **5**
5. Data Backup and Recovery **5**
6. Error Handling and Logging **5**
7. Image Compression and Storage Feasibility **6**
6.1 Overview **6**
6.2 Image Compression Analysis **6**
6.2.1 Average File Size for Source Images **6**
6.2.2 JPEG XL Compression Efficiency **6**
Compression Estimates: **6**
6.3 Total Storage Requirement Calculations **7**
6.3.1 Storage Requirements for Images **7**
6.3.2 Storage Requirements for Text Data **7**
6.3.3 Combined Storage Estimates **7**
6.4 Conclusion and Recommendations **7**
6.5 Next Steps **7**
6.6 Implementation Notes **8**
Libraries and Tools: **8**
Considerations for Color Data: **8**
Fallback Options: **8**
8. Possible Error Scenarios **8**
1. Database Connection Errors 8
2. Data Retrieval Errors 8
3. Data Update and Insert Errors 8

4. Data Synchronization Issues 9
5. API Errors 9
6. Security-Related Errors 9
7. Storage and Cost Management Errors 9
8. Client-Side Application Errors 9
9. Logging and Monitoring Failures 9

1. Updated Database Schema Design

The application will store all recipe-related data in an Azure SQL Database. The updated schema follows industry best practices, using normalized tables to reduce redundancy, maintain data integrity, and facilitate efficient querying.

Ingredient Reference Table

Purpose: To store unique, standardized ingredient names and details.

- Fields:
    - IngredientRefID (Primary Key): Unique identifier for each ingredient.
    - IngredientName: Standardized name of the ingredient (e.g., 'brown onion').
    - IngredientCategory (Optional): Category of the ingredient (e.g., 'Onion').
    - NormalizedIngredientName: Normalized name for consistency checks (e.g., all lowercase, stripped of spaces).
    - UnitType (Optional): Default unit of measurement (e.g., 'grams', 'ml').

Recipes Table

Purpose: To store basic information about each recipe.

- Fields:
    - RecipeID (Primary Key): Unique identifier for each recipe.
    - RecipeName: Name of the recipe.
    - Description: Brief description of the recipe.
    - DateCreated: Timestamp of when the recipe was created.
    - DateModified: Timestamp of when the recipe was last modified.
    - ServingSize: Number of servings.
    - Tags: Tags for dietary preferences, allergens, etc.

RecipeIngredients Table (Junction Table)

Purpose: To manage the many-to-many relationship between recipes and ingredients.

- Fields:
    - RecipeIngredientID (Primary Key): Unique identifier for each record.
    - RecipeID (Foreign Key): Reference to the associated recipe.
    - IngredientRefID (Foreign Key): Reference to the associated ingredient from the Ingredient Reference Table.
    - Quantity: Amount of the ingredient needed for the recipe.
    - Unit: Unit of measurement (e.g., grams, cups).

Instructions Table

Purpose: To store detailed step-by-step instructions for each recipe.

- Fields:

- InstructionID (Primary Key): Unique identifier for each instruction step.
- RecipeID (Foreign Key): Reference to the associated recipe.
- StepNumber: Order of the step in the recipe.
- InstructionText: Text of the instruction.
- StepType: Type of instruction (e.g., Prep, Cooking, Presentation).

Images Table

Purpose: To store references to recipe images.

- Fields:

  - ImageID (Primary Key): Unique identifier for each image.
  - RecipeID (Foreign Key): Reference to the associated recipe.
  - ImageURL: URL where the image is stored in Azure Blob Storage.
  - DateUploaded: Timestamp of when the image was uploaded.

2. Updated Approach for Ingredient Management

The application employs a standardized approach for managing ingredients with fuzzy matching to reduce duplication and ensure data consistency.

2.1 Ingredient Standardization

All ingredient entries are automatically standardized upon input by converting text to lowercase, trimming whitespace, and removing special characters. This ensures consistency in the Ingredient Reference Table.

2.2 Fuzzy Matching with User Prompt

After standardization, the app performs a fuzzy matching check against existing entries in the Ingredient Reference Table. If a similar entry is found within a defined threshold, the user is prompted to confirm or correct the entry. If no match is found, the ingredient is added directly. If an exact match is found, the ingredient is added without prompting.

3. API Integration for Nutritional Analysis

API: Spoonacular API for calculating nutritional information.

Endpoint: /recipes/parseIngredients

Method: POST

- Request Format: Send a structured JSON payload containing the list of ingredients.

Example Payload:

```
{
  "ingredients": [
```

```
  {"name": "chicken breast", "quantity": 200, "unit": "grams"},
  {"name": "olive oil", "quantity": 2, "unit": "tablespoons"}
 ]
}
```

- Response Handling:
    - Parse the JSON response to extract calorie, fat, sugar, and other nutritional values.
    - Store the calculated values in the Dietary Information Table.
- Error Handling:
    - Implement retry logic with exponential backoff for API call failures.
    - Log errors with detailed information for debugging.

4. Data Synchronization Strategy

Initial Data Load: On application startup, check for updates and download only the required recipe data (excluding images) from the Azure SQL Database to the local device.

Lazy Loading for Images: Images will only be fetched from Azure Blob Storage when a specific recipe is accessed to minimize data transfer and storage costs.

Periodic Synchronization: Implement a periodic synchronization routine that checks for updates at regular intervals or when certain actions occur (e.g., adding or editing a recipe).

Use timestamp-based comparison to identify changes and avoid unnecessary data transfers.

5. Data Backup and Recovery

Backup Configuration: Use Azure SQL's automatic backup functionality to create weekly backups.

Retention Policy: Maintain a rolling backup for the last 5 weeks, automatically deleting older backups to manage storage.

Backup Exclusions: Configure backups to exclude image files, focusing only on critical recipe data to save storage space and costs.

6. Error Handling and Logging

- Error Handling:
    - Implement comprehensive error handling for all data operations (e.g., database access, API calls).

    - Use transaction management to ensure data integrity during CRUD operations. Rollback transactions on failure.
    - Handle specific error scenarios like network issues, authentication failures, and data inconsistencies.

- Logging Strategy:

    - Utilize a logging framework such as Serilog or NLog to capture errors, warnings, and important system events.
    - Configure log storage to an Azure log service or local files for easier access and monitoring.

7. Image Compression and Storage Feasibility

7.1 Overview

This section explores the feasibility of storing 200 recipe images along with text data within a 5GB storage limit using JPEG XL compression. The goal is to ensure that the application remains scalable while maintaining a balance between image quality and storage efficiency.

7.2 Image Compression Analysis

7.2.1 Average File Size for Source Images

A standard photo taken by the Google Pixel 5 camera (12MP, 4000 x 3000 pixels) in JPEG format ranges between 3MB and 6MB. The variation in size depends on the image's complexity, color range, and level of detail.

7.2.2 JPEG XL Compression Efficiency

JPEG XL is a modern image format offering both lossless and lossy compression, providing better compression ratios than older formats like JPEG and WebP.

Compression Estimates:

- Lossless Compression:

    - Typically achieves a 20-40% reduction in file size.
    - Estimated size per image after compression: 1.8MB to 4.8MB.

- Lossy Compression:

    - Achieves a more significant reduction of 60-80% while maintaining good visual quality.
    - Estimated size per image after compression: 0.6MB to 2.4MB.

7.3 Total Storage Requirement Calculations

7.3.1 Storage Requirements for Images

- For 200 images:
    - Lossless Compression:
        - Minimum Estimate: 200 x 1.8MB = 360MB
        - Maximum Estimate: 200 x 4.8MB = 960MB
    - Lossy Compression:
        - Minimum Estimate: 200 x 0.6MB = 120MB
        - Maximum Estimate: 200 x 2.4MB = 480MB

7.3.2 Storage Requirements for Text Data

Text data per recipe (title, ingredients, steps, tags, etc.) is estimated at around 10KB.

For 200 recipes: Total text data: 200 x 10KB = 2MB

7.3.3 Combined Storage Estimates

- Lossless Compression:
    - Total Minimum: 362MB (360MB images + 2MB text)
    - Total Maximum: 962MB (960MB images + 2MB text)
- Lossy Compression:
    - Total Minimum: 122MB (120MB images + 2MB text)
    - Total Maximum: 482MB (480MB images + 2MB text)

7.4 Conclusion and Recommendations

- Feasibility:
    - Both lossless and lossy JPEG XL compression methods allow for storing 200 images and associated recipe data well within the 5GB storage limit.
- Recommended Strategy:
    - Lossless Compression is advisable if image quality is a priority, as it maintains full image fidelity while still offering significant storage savings.
    - If maximizing storage capacity is more critical, Lossy Compression should be considered, as it provides a higher compression ratio while retaining acceptable visual quality.

7.5 Next Steps

- Implement the JPEG XL compression using available libraries (libjxl or tools like cjxl) in the chosen development environment.

- Conduct testing with sample images to verify compression effectiveness and fine-tune parameters for optimal quality and storage balance.

7.6 Implementation Notes

Libraries and Tools:

- JPEG XL Libraries: Utilize libjxl for cross-platform compatibility.
- Alternative Tools: Consider ImageMagick, which supports JPEG XL, for additional image processing capabilities.

Considerations for Color Data:

- Higher compression rates may impact color fidelity; testing should include diverse images to evaluate acceptable quality levels.

Fallback Options:

- If JPEG XL support proves inadequate or compatibility issues arise, WebP remains a viable fallback option with broader platform support.

8. Possible Error Scenarios

A detailed listing of potential error scenarios affecting the project, database, and client-side interactions has been added for reference and future planning.

1. Database Connection Errors

- Timeout: The application fails to establish a connection with the Azure SQL Database within the specified time limit.
- Network Issues: Loss of network connectivity between the app and the database server could cause connection failures.
- Authentication Failures: Incorrect or expired credentials when attempting to connect to the Azure SQL Database.

2. Data Retrieval Errors

- Data Not Found: Attempting to retrieve a recipe or other data that does not exist in the database.
- Data Format Errors: Mismatch between the expected data format (e.g., JSON, XML) and the format returned by the database.
- Invalid Query Syntax: Malformed SQL queries sent to the database, causing query execution to fail.

3. Data Update and Insert Errors

- Duplicate Entry: Attempting to insert data that violates unique constraints (e.g., adding a recipe with a duplicate name).
- Constraint Violations: Violating database constraints such as foreign keys, unique constraints, or check constraints.

- Transaction Failures: Errors occurring during a transaction (e.g., inserting multiple records where one fails), leading to incomplete or partial updates.

4. Data Synchronization Issues

- Conflicts: Conflicts between the local data on the device and the data in the cloud database during synchronization.
- Partial Data Transfer: Incomplete data transfer due to network issues or interruptions during synchronization.

5. API Errors

- Rate Limit Exceeded: Exceeding the predefined rate limit for API requests, causing the server to reject further requests temporarily.
- Invalid API Requests: Malformed requests sent to the API (e.g., missing parameters, incorrect HTTP methods).
- Server-Side Errors: 500-series errors (e.g., Internal Server Error, Service Unavailable) due to backend issues or unexpected server failures.

6. Security-Related Errors

- Unauthorized Access: Unauthorized attempts to access the API or database due to incorrect or missing authentication.
- Data Leakage: Potential exposure of sensitive data if encryption or HTTPS is not properly implemented.

7. Storage and Cost Management Errors

- Exceeded Storage Quota: Exceeding the 5GB storage limit in Azure, potentially incurring additional costs.
- Early Deletion Fees: Deleting data before the minimum storage duration, leading to additional charges.

8. Client-Side Application Errors

- UI Errors: Errors in rendering the user interface, such as missing elements, unresponsive buttons, or display issues.
- App Crashes: Application crashes due to unhandled exceptions, memory leaks, or excessive resource consumption.
- Compatibility Issues: Incompatibility with certain devices or operating system versions, leading to unexpected behavior.

9. Logging and Monitoring Failures

- Logging Failures: Errors in the logging framework, such as failure to write logs to a file or database.
- Insufficient Error Data: Missing or incomplete error information in logs, making debugging difficult.