Logan Dawes – logan.dawes@okstate.edu - CS-4983

Project Documentation

MealForge - Meal Prep Optimizer Application

Resources :

GitHub Repo: https://github.com/LoganDawes/MealForge

DB Server Name: mealforge-db.mysql.database.azure.com

Problem Statement

Maintaining and balancing a nutritional diet is a huge challenge for health-conscious people who want to improve their well-being and find meals that align with their personal diets. Many meal-planning apps can fail to consider the specific nutritional requirements that go into each recipe and can lead to extended periods of research for something that should be simple.

My proposed solution to this problem is a Meal prep optimization app to offer recipes based on user-input ingredients and dietary restrictions. Both ingredients and recipes can be personalized with filters based on nutritional value, dietary preferences, and health benefits. Users can save ingredients that match with their diet to further personalize the recipes offered to them.

This is a fairly significant problem for individuals who care about their nutritional health and personal diet or have certain allergies that restrict them from eating specific meals. These people should not have to struggle with finding a solution in order to be healthy.

The scope of this problem covers a wide margin of the population that takes their eating habits seriously and even includes those who are beginning to diet for its benefits and find the process tedious to accomplish.

Timeline

Week 1 : Jan 26 – Feb 01:

Identify functional & non-functional requirements for the application, document system requirements into well-detailed report.

Identify nutritional resources for the application to use for suggestions.

[Deadline: Problem Statement and Initial Timeline : Jan 31]

Week 2 : Feb 02 – Feb 08:

Develop high-level architectural design to outline the overall structure of the system. Provide UML diagrams for components, structure, behavior, and interactions.

Week 3 : Feb 09 – Feb 15:

Complete UML diagrams: study the microservices architecture and how it would be implemented.

[Deadline: System Requirements : Feb 14]

Logan Dawes – logan.dawes@okstate.edu - CS-4983

Week 4 : Feb 16 – Feb 22:

Implementation: Set up GitHub repository, research into developing microservices architecture using Django backend.

Week 5 : Feb 23 – Mar 01:

Implementation: Backend: Initialization

Created each service as an individual app and worked on containerizing them using Docker.

Week 6 : Mar 02 – Mar 08:

Implementation: Backend: User registration

Implemented the basic flow from API gateway to registering a user, as well as login/logout. Created a cloud server with Azure cloud for a MySQL database. Connected database for user creation.

[Deadline: Architectural Design & System Modeling : Mar 7]

Week 7 : Mar 09 – Mar 15:

Implementation: Backend: API calls

Added backend API calls related to getting user preferences, and getting ingredient and recipe details.

Week 8 : Mar 16 – Mar 22:

Implementation: Backend: Saved Collections

Added ability for users to save ingredients and recipes, cache and database refreshing management, and started on logging service.

Week 9 : Mar 23 – Mar 29:

Implementation: Backend: Finalizing

Finalized backend API calls and logging service. Started initializing frontend files and containerizing.

Week 10 : Mar 30 – Apr 05:

Implementation: Frontend: Routing

Implemented routing to different frontend pages and tested connection to backend. Started on component implementation.

Week 11 : Apr 06 – Apr 12:

Implementation: Frontend: Backend calls part 1

Implemented the rest of the basic components, added backend functionality for registration, preferences, and search.

Week 12 : Apr 13 – Apr 19:

Implementation: Frontend: Backend calls part 2

Finalized basic frontend setup, began testing all components and polished out some bugs.

Week 13 : Apr 20 – Apr 26:

Testing: Final system testing, iron out bugs

Through testing, fixed some gaps in logic such as proper error messages, image/text placements, parameter passing, etc.

[Artificial Deadline for tested & deployed implementation]

Week 14 : Apr 27 – May 02:

Creating presentation slides and demo video.

20 min Final Project Presentation: showcase project lifecycle and include demonstrations.

[Deadline: Final Project Presentation : May 01]

Functional requirements

User Management

- The system shall allow Users to sign up to create their own account, with unique identifying information.
- Users shall be able to log in, log out, and switch accounts.
- The system shall securely store passwords using encryption.
- Users shall be able to customize their preferences including name, diet, intolerances, calories, and nutrients.
- The system shall validate the user's email and password format before creating an account.
- Users shall be able to recover their password with a password recovery method.
- The system shall allow administrators to manage registered users and not reveal their password.

Recipe Management

- Each recipe shall include breakdown of dietary groups, nutrients, calories, intolerances, and other health benefits.
- Each recipe shall provide a link to the original recipe page.
- Users shall be able to search for recipes by name, category, and/or tag.
- The system's recipe search functionality shall be sortable and filterable based on diet, intolerances, calories, and nutritional value.
- Each ingredient shall have a link provided to the ingredient's information page.
- Users shall be able to save recipes on their personal recipe book.

Ingredient Management

- Each ingredient shall include breakdown of dietary groups, nutrients, calories, intolerances, and other health benefits.
- Users shall be able to search for ingredients by name, category, and/or tag.
- The system's ingredient search functionality shall be sortable and filterable based on diet, intolerances, calories, and nutritional value.
- The system shall dynamically update the recipe list based on available ingredients.

- Users shall be able to add ingredients to their personal collection of ingredients for use in recipes.

Personalization

- The system shall implement a search algorithm that reflects the user’s preferences.
- The system shall provide personalized health benefits for each recipe based on the user’s preferences.

Non-functional requirements

Scalability

- The system shall allow new features and modules to integrate easily.
- The system shall have proper API integration for ingredients and recipes.
- The system’s database shall be able to expand without degradation of performance.

Security

- User data shall be secure and stored with encryption.
- The system shall prevent unauthorized access through multiple protocols.
- The system shall properly audit user activity.
- The system shall provide multi-factor authentication options.

Maintainability

- The system shall include proper documentation of the system and code provided to simplify development.
- The system shall incorporate a modular, microservices design to ensure quick updating of features.
- The system shall support version control to track updates in the application.

Reliability

- The system shall have proper error handling and provide easy debugging.
- The system shall have constant up time, and any downtime shall be communicated to users.
- The system shall save and keep track of the user’s input data across instances.

Performance

- The system shall respond to queries in less than 2 seconds.
- The system shall handle 1,000 concurrent users and sudden spikes in activity.

Usability

- The system’s UI shall be adaptable to multiple screen resolutions and can open on a mobile device.
- The system’s UI shall allow any key action to be performed within 3 clicks.
- The system shall include clear contrast of color and formatting to indicate nutritional aspects.

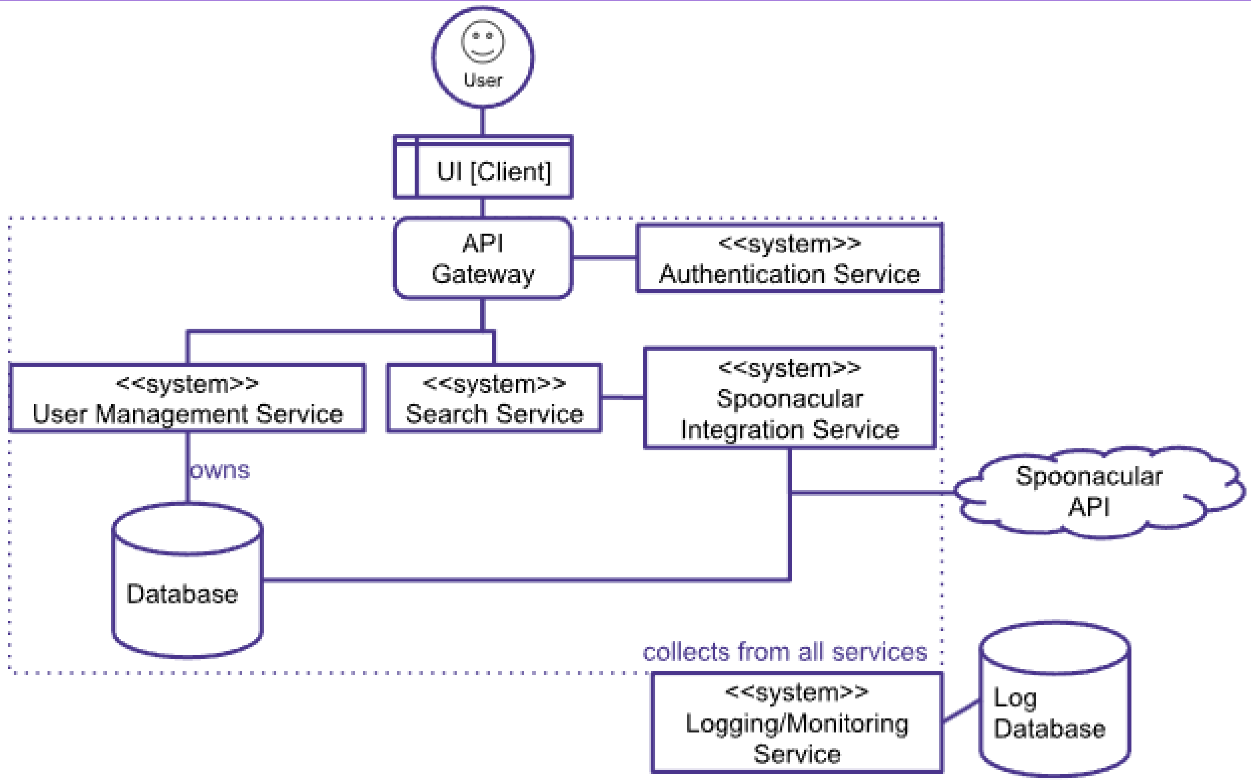
Assumptions

- Users have internet access to interact with the system.
- Users have a web browser that is not outdated.
- Users have access to an active email account.
- External APIs used by the system will remain supported and function as expected.

Constraints

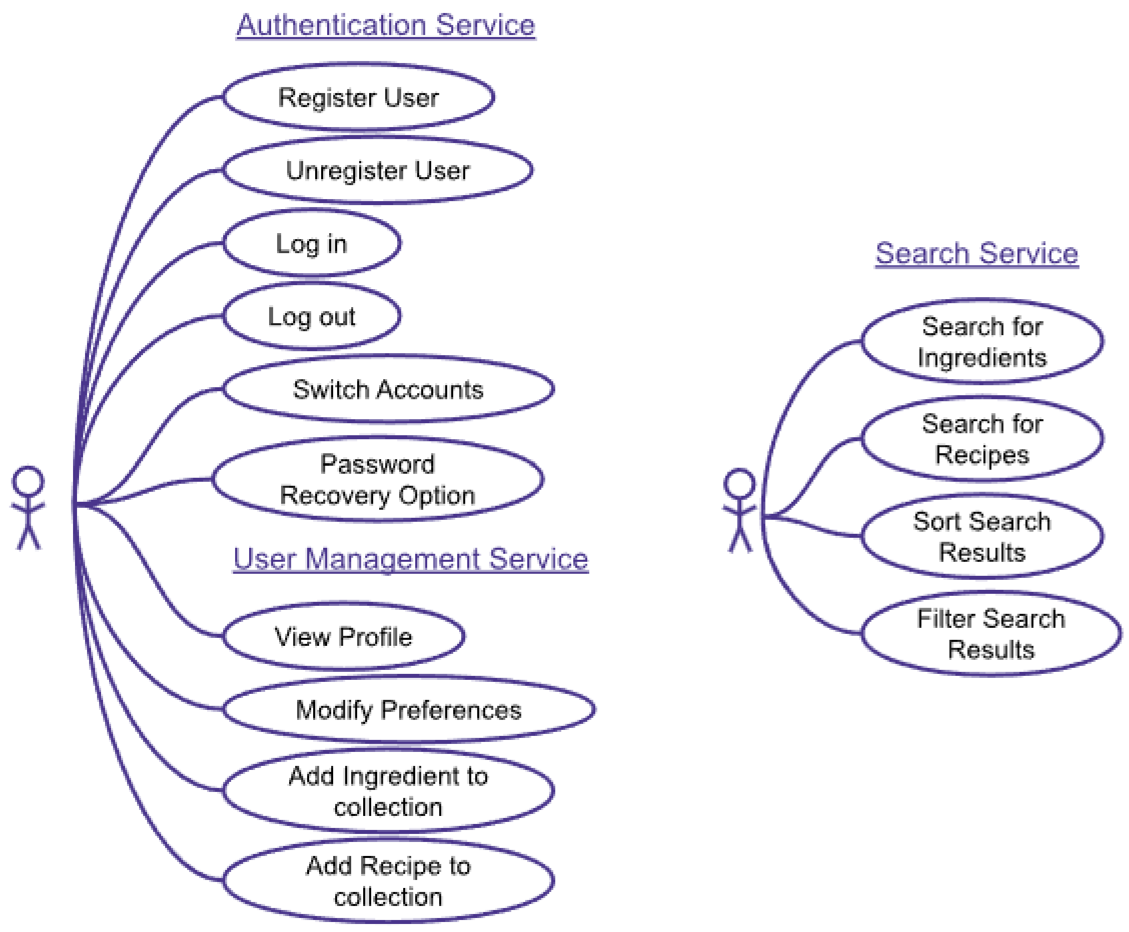
- The system shall comply with data protection regulations when handling user data.
- The system shall comply with the terms of use outlined by any third-party API.
- The system shall adhere to the limitation of third-party APIs for ingredients and recipes.

Architectural Design


User
UI [Client]
API Gateway
<<system>> Authentication Service
<<system>> User Management Service
<<system>> Search Service
<<system>> Spoonacular Integration Service
owns
Database
Spoonacular API
collects from all services
<<system>> Logging/Monitoring Service
Log Database

System Modeling

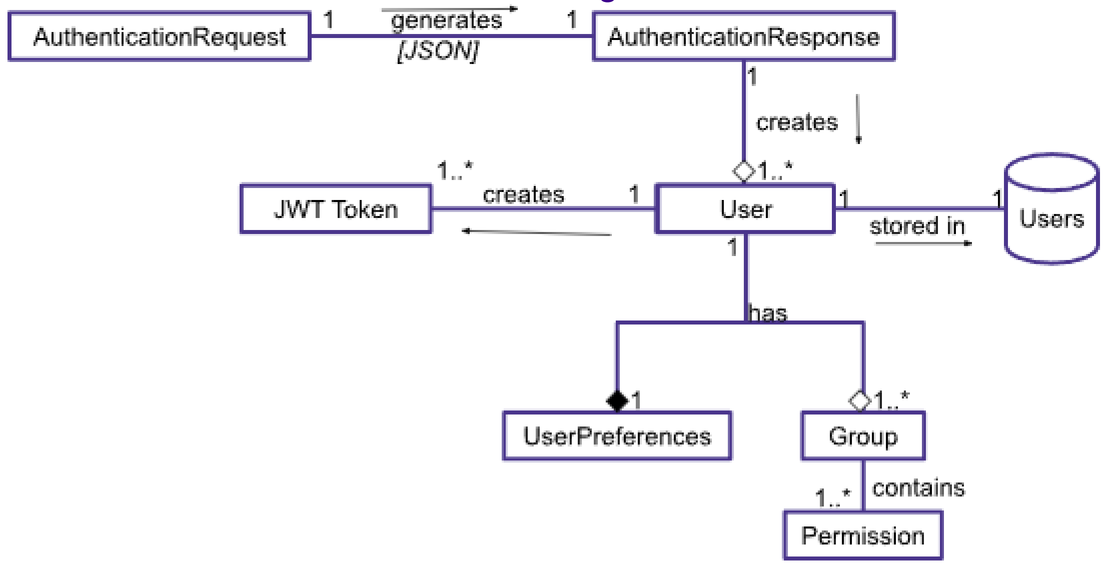
Use Case Diagram (User requirements)



Logan Dawes – logan.dawes@okstate.edu - CS-4983

Structural Diagrams:

Authentication Service & User Management Service



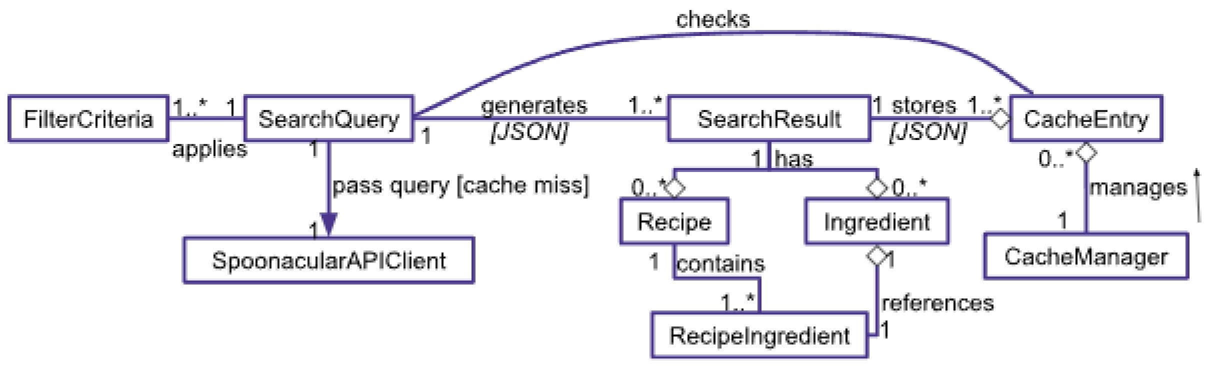
Authentication:

When user registers, an *AuthenticationRequest* is sent to the Authentication Service to generate an *AuthenticationResponse*, creating a *User* and JWT *Token*.

User Management:

Each *User* is linked to their own *UserPreferences* and a *Group* containing *Permissions* for authorization. Each User and their saved recipes is stored in the database.

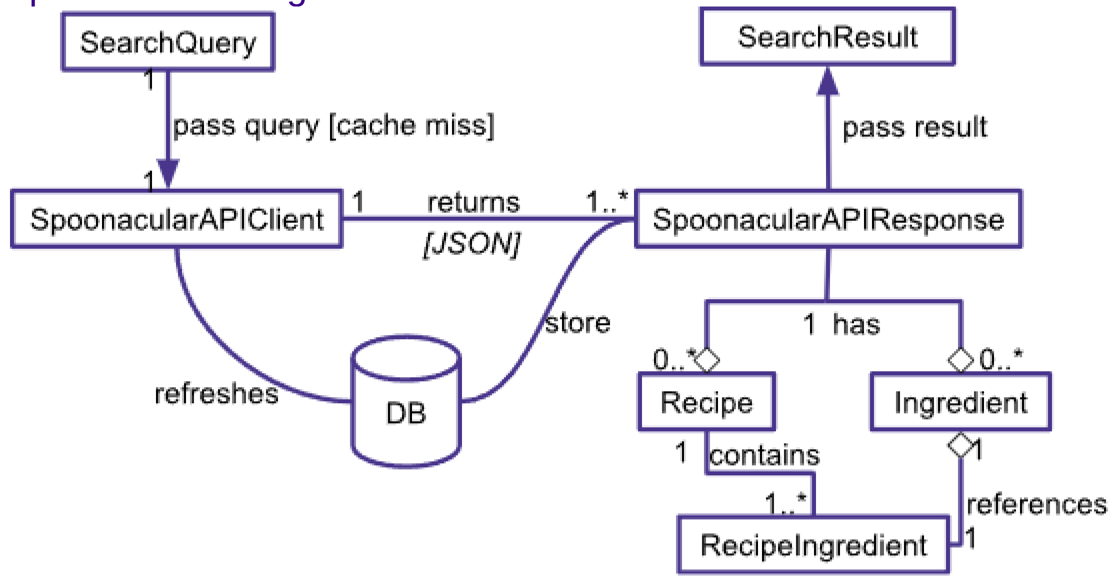
Search Service



User submits a *SearchQuery* including *FilterCriteria*. *SearchQuery* searches Cache, if hit: generates *SearchResult* from cache, if miss: sends query to *SpoonacularAPIClient*. *SearchResult* contains *Recipes* and *Ingredients*, and stores result in Cache for 1 hour, tracked by *CacheManager.*

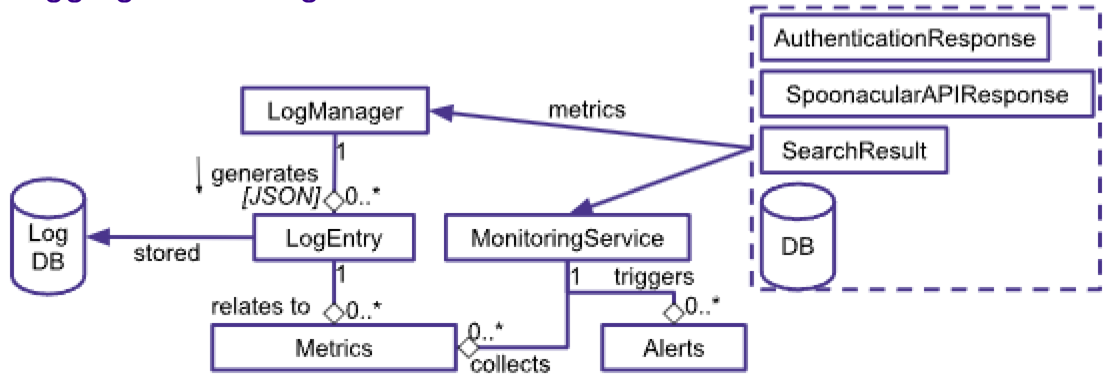
Logan Dawes – logan.dawes@okstate.edu - CS-4983

Spoonacular Integration Service



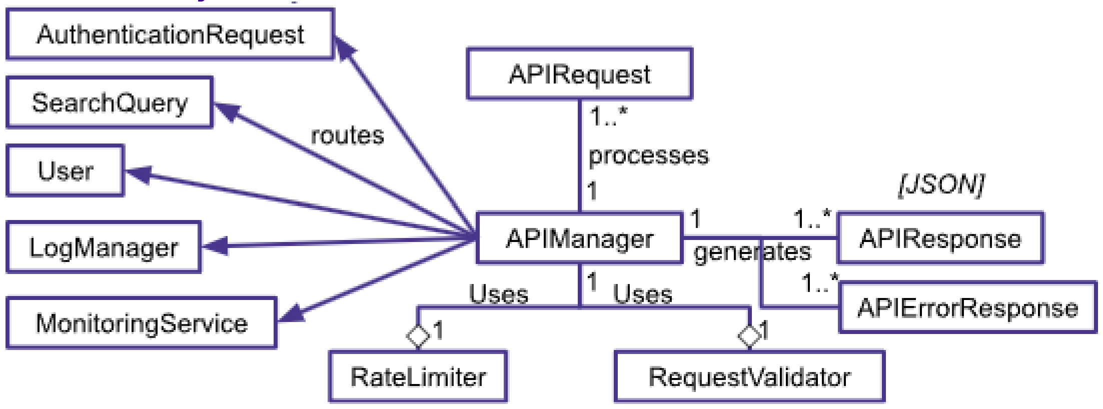
SpoonacularAPIClient sends an API call to Spoonacular based on *SearchQuery*, or entries missing from *Database*, retrieves *SpoonacularAPIResponse* of requested *Recipes* and *Ingredients*. The response is used for the Search Result to query or to refresh the requested database entries.

Logging/Monitoring Service



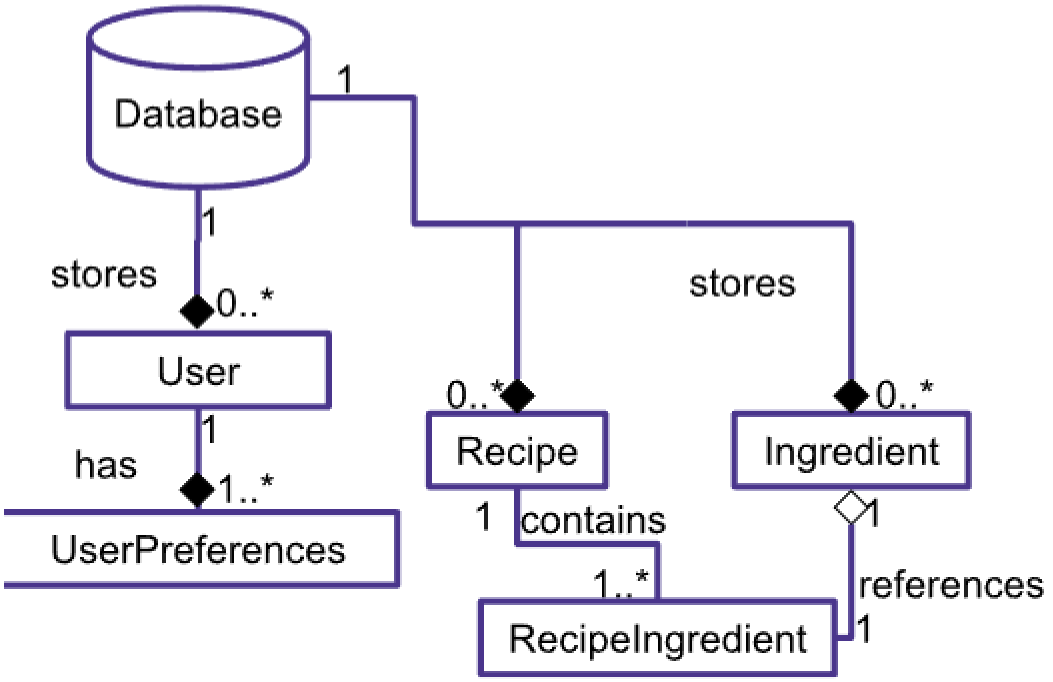
Other services generate logs for user activity, authentication attempts, API calls, etc. sent to *LogManager*, which generates *LogEntry*. *MonitoringService* processes *Metrics* to detect system issues and triggers *Alerts.*

API Gateway



APIManager receives *APIRequests* from the client and validates it using *RequestValidator*. Then it checks the *RateLimiter*, before routing the request to appropriate microservice. *APIManager* formats response from microservice into *APIResponse* to send back to client. If unsuccessful, sends *APIErrorResponse*.

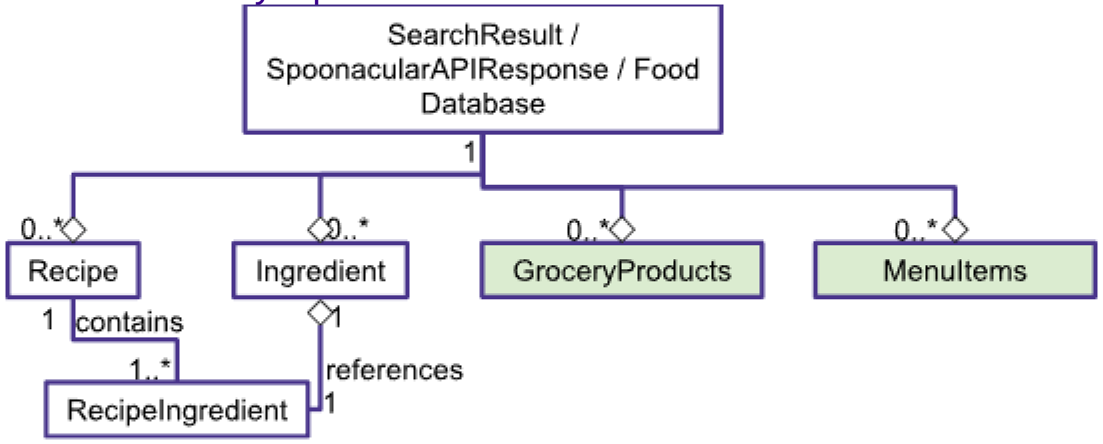
Databases



The *Database* stores *Users* created by User Management Service to be retrieved later, and stores *Recipe*s and *Ingredient*s saved by the user, contents need to be updated by Spoonacular Integration Service every 24 hours, managed by SpoonacularAPIClient.

Logan Dawes – logan.dawes@okstate.edu - CS-4983

Planned Scalability Options



GroceryProducts and *MenuItems* can also be requested from Spoonacular API to be searched and saved by the user in a personal collection.