CS443 CONTENTA APP
3-PAGER PROJECT DESCRIPTION

Group E

Members:
Onurcan Ataç
Ömer Oktay Gültekin
Kutay Şenyiğit

Problem Introduction:

Buying packaged products is commonly routine for some households; however, with increasing awareness, there comes a need to know what is within the products customers consume. The necessity even increases while in a foreign country. There are ingredients that a less informed customer may not have the necessary detailed knowledge about. Informed customers are tempted to buy a product after considering all the contents they know and skipping the uncommonly named ones, while the others do not even do that much.

Purpose of Contenta:

The main goal of Contenta is to give users fast feedback on the ingredients within their packaged products on the go. Within the app, consumers may scan the ingredients of packaged foods to obtain detailed information about the ingredients, especially any health concerns. The application seeks to read, detect, and list the ingredients listed on the packages in an easily interactable format. Moreover, it warns and informs them based on their concerns, risks, and backgrounds. The app Contenta is, therefore, designed to provide customers with quick and efficient information, aiming to assist them in their daily product consumption by offering transparency about the contents of what they consume. Contenta uses image recognition algorithms to detect allergens, harmful ingredients, and nutrition facts to calculate an EU-standard Nutri-Score from product labels. Also, preapproved expert users can post blogs to educate the users about ingredients.

Target Market:

The primary target market of Contenta consists of health-conscious consumers, allergy sufferers, parents who are mindful of what their children eat, and vegans and vegetarians. Contenta is an application that any consumer can use on their mobile device.

Key Metrics and Technology Stack:

The retention rate of the users will measure the success of the project. Correctness of the text recognition and extraction will be the critical factor for a high retention rate.
Contenta was developed using a Flutter frontend, with all backend services hosted on AWS.

Potential Risks:

There is a risk of poor ingredient recognition in the project. This risk will show up when the quality of the extracted text is poor, or the database doesn't have the ingredient. If possible, the risks will be addressed using NLP algorithms to correct the mistakes. Otherwise, the request will be made to retake the photo. Also, databases will be reviewed periodically with the latest data.

Planned Cost:

The result of the cost calculation for 1000 monthly active users using AWS Cost Calculator is visible at https://calculator.aws/#/estimate?id=a71e9d3b7ac9486e7d3972f44c81fc7e984c138d.

Teamwork Details:

In our team, we all knew each other before starting the project, which made collaboration smoother. We preferred meeting in person because it enhanced our communication and teamwork. We were committed to supporting each other in individual tasks and whatever the team needed. We regularly update each other on our progress, sharing the responsibility so that everyone is in the loop and can pitch in when necessary. This approach kept us aligned and prepared to tackle any issues together.

Architectural Choices:

a) Design Choice:

We implemented Contenta with a serverless architecture since it was the best fit according to our requirements. There is no need to have a running machine instance all the time, and our functions are guaranteed to run at most 15 minutes.

Our main workflow includes a step function comprising a chain of Lambda functions. The core features of our application are implemented in this step function. The workflow consists of a lambda function that uploads the uploaded image into s3 after converting it into the gray-scale image. Then, two parallel states are started to execute. One extracts the ingredients and allergens from the image and searches them in our databases. The other one is for extracting the nutrition values of the product and calculating the nutriscore value.

b) Databases:

We used multiple types of databases concerning their functionalities. We utilized RDS for the data that needs complex queries, DocumentDB for in-text document searches, and DynamoDB for key-value type searches. We didn't use Aurora. Instead, we used PostgreSQL since Aurora has no free tier option.

c) Security Measures:

- When designing our application, we also considered meeting security concerns. Therefore, we created IAM users to manage the authorization permissions in our AWS account. We used Amazon Cognito to create user roles in our application. While implementing those, we considered the Least Privilege Principle. Also, we utilized the IAM Access Analyzer to find the possible unused access permissions in the IAM roles. In addition, security groups are arranged accordingly.

- We utilized CloudWatch to keep our application logs in case any security issues occur so that we can check our logs to gain more information about the issue. We also created custom metrics in CloudWatch that trigger an alarm in case of step-machine execution failure.

- We ensured that our databases reside on private networks since we wanted to reduce exposure to the internet and decrease the risk of unauthorized access. Their connection strings are stored in environment variables, while the username and passwords come from the secret manager.

- We integrate the Amazon WAF service with the cognito for a safer user sign-in experience. Also, we added an authorization step for the API endpoints that Cognito verifier verifies the incoming token if it is created by it and valid.

d) Other Services Used:

We utilized Amazon Textract to extract the table and text from the images of the ingredients the user took. This text was processed by the Command R+ LLM model and returned in JSON form to perform the necessary search processes later. We could have used Amazon Bedrock to access the Command R+ model, but since it is already free, we have decided to make API calls to integrate with the model directly.

Since the admins maintain the database of the ingredients and allergens, we have created a notification service that when an ingredient is extracted from the text but not found in the database, we notify the users in Cognito admin group via e-mail about the missing ingredients in the database to make it possible for them to take action if they want. This way, the more the application is used, the higher the quality of its feedback will increase.

e) Contenta Architecture Diagram:

The complete diagram can be seen from the following link: https://drive.google.com/file/d/19RACB9zMb3MOQpcbFVFnlm4N4XO3eq-x/view?usp=sharing.

Lessons Learned:

- In our initial project design, we included services such as Amazon Comprehend and Amazon Translate. However, as the project progressed, we realized we could utilize the Command R+ LLM model to take the role of those services. We tried to use them, but their performance was unsatisfactory in our use case.

- Our initial design did not involve Amazon SNS and alarms through CloudWatch metrics. We incorporated those services into our step function. Later, we concluded that using the SNS and alarms was very beneficial since we got notified whenever our application threw an error or had missing data. Therefore, we could diagnose the problems, troubleshoot, and take action immediately.

- We could have used DynamoDB instead of DocumentDB, but we didn't change it later since it was working and its performance was in the range of our expectations.