Receipt Processor API Documentation

Atharva Deshpande

1 Introduction

The Receipt Processor API is a web service that processes JSON receipts and awards points based on predefined rules. This document provides a detailed description of the API endpoints, request and response formats, rules for awarding points, and instructions for running the application.

2 API Endpoints

The Receipt Processor API has two endpoints:

1. **/receipts/process (POST)**: This endpoint takes a JSON receipt as input and returns a JSON object with an ID for the receipt. The ID can be used to retrieve the number of points awarded for the receipt.
2. **/receipts/{id}/points (GET)**: This endpoint takes the ID of a processed receipt as input and returns a JSON object containing the number of points awarded for that receipt.

3 Payload Format

The payload format for the JSON receipt is as follows:

```
{
  "retailer": "Retailer Name",
  "total": "Total Amount",
  "items": [
    {
      "shortDescription": "Item Description 1",
      "price": "Item Price 1"
    },
    {
      "shortDescription": "Item Description 2",
      "price": "Item Price 2"
    },
    ...
  ],
  "purchaseDate": "Purchase Date (YYYY-MM-DD)",
  "purchaseTime": "Purchase Time (HH:MM)"
}
```

4 Response Format

The response format for the ID and points JSON objects is as follows:

```
{
  "id": "Generated ID for the receipt"
}
```

```
{
  "points": "Number of points awarded"
}
```

5 Rules for Awarding Points

The API follows these rules to award points for a receipt:

1. One point for every alphanumeric character in the retailer name.
2. 50 points if the total is a round dollar amount with no cents.
3. 25 points if the total is a multiple of 0.25.
4. 5 points for every two items on the receipt.
5. If the trimmed length of the item description is a multiple of 3, multiply the price by 0.2 and round up to the nearest integer. The result is the number of points earned.
6. 6 points if the day in the purchase date is odd.
7. 10 points if the time of purchase is after 2:00pm and before 4:00pm.

6 Running the Application

To run the Receipt Processor API, follow the instructions provided in the README.md file of the GitHub repository. Ensure you have the required dependencies installed, such as Go and Docker.

7 Unit Testing

The application includes unit tests to verify the correctness of the solution. Test cases cover different scenarios, including valid inputs, invalid inputs, and edge cases.

8 Error Handling and Validation

The application has appropriate error handling for different scenarios. It returns HTTP status codes along with error messages or codes to provide detailed information about encountered errors. Input validation is implemented to ensure all required fields are present and have valid values.

9 Performance Optimization

Depending onthe expected usage and scale of the Receipt Processor API, certain performance optimizations can be considered. Here are a few suggestions:

1. Utilize efficient algorithms and data structures: Evaluate the algorithms used for calculations and data processing. Look for opportunities to optimize them for better performance. Consider using more efficient data structures, such as hash maps or balanced trees, to improve lookup and retrieval operations.
2. Implement caching: If there are repeated calculations or data retrievals that remain constant for a certain period, consider implementing a caching mechanism. This can help avoid redundant computations and improve response times.
3. Optimize database queries: If the API interacts with a database, review the database queries and optimize them for better performance. Ensure that proper indexes are in place, and consider techniques like query optimization and database tuning.
4. Enable parallel processing: If there are independent parts of the code that can be executed in parallel, consider utilizing concurrency techniques such as goroutines in Go to improve the overall processing speed.
5. Monitor and analyze performance: Implement monitoring and logging mechanisms to track the performance of the API in real-time. Analyze the collected data to identify bottlenecks and areas for improvement. This can help fine-tune the application and make informed decisions on further performance optimizations.

Remember, the choice of performance optimizations depends on the specific requirements and constraints of your application. It's essential to profile the application, identify the areas that require improvement, and carefully test any changes to ensure they deliver the expected performance enhancements.