**Solution**

**Project Structure**

receipt-processor/

│

├── main.go # The main application file where the server is started

├── handler.go # Handlers for the API endpoints

├── receipt.go # Defines the Receipt struct and methods for points calculation

└── store.go # In-memory store for receipts

**Step 1: Define the Receipt Struct and Points Calculation**

// receipt.go

package main

import (

"math"

"strings"

"time"

)

// Receipt represents a single receipt and its items

type Receipt struct {

ID string `json:"id"`

Retailer string `json:"retailer"`

PurchaseDate string `json:"purchaseDate"`

PurchaseTime string `json:"purchaseTime"`

Items []Item `json:"items"`

Total float64 `json:"total"`

PointsAwarded int `json:"-"`

}

// Item represents a single item in the receipt

type Item struct {

ShortDescription string `json:"shortDescription"`

Price float64 `json:"price"`

}

// CalculatePoints calculates the total points for the receipt

func (r \*Receipt) CalculatePoints() {

r.PointsAwarded = len(strings.TrimSpace(r.Retailer))

// Rule: 50 points if the total is a round dollar amount with no cents.

if r.Total == math.Floor(r.Total) {

r.PointsAwarded += 50

}

// Rule: 25 points if the total is a multiple of 0.25.

if math.Mod(r.Total\*100, 25) == 0 {

r.PointsAwarded += 25

}

// Rule: 5 points for every two items on the receipt.

r.PointsAwarded += (len(r.Items) / 2) \* 5

for \_, item := range r.Items {

trimmedDesc := strings.TrimSpace(item.ShortDescription)

if len(trimmedDesc)%3 == 0 {

pointsFromPrice := math.Ceil(item.Price \* 0.2)

r.PointsAwarded += int(pointsFromPrice)

}

}

date, \_ := time.Parse("2006-01-02", r.PurchaseDate)

// Rule: 6 points if the day in the purchase date is odd.

if date.Day()%2 != 0 {

r.PointsAwarded += 6

}

// Rule: 10 points if the time of purchase is after 2:00pm and before 4:00pm.

purchaseTime, \_ := time.Parse("15:04", r.PurchaseTime)

if purchaseTime.Hour() >= 14 && purchaseTime.Hour() < 16 {

r.PointsAwarded += 10

}

}

**Step 2: In-memory Store for Receipts**

// store.go

package main

import "sync"

// Store holds the receipts in-memory

type Store struct {

receipts map[string]\*Receipt

mu sync.Mutex

}

// NewStore creates a new store

func NewStore() \*Store {

return &Store{

receipts: make(map[string]\*Receipt),

}

}

// SaveReceipt saves a receipt in the store

func (s \*Store) SaveReceipt(receipt \*Receipt) {

s.mu.Lock()

defer s.mu.Unlock()

s.receipts[receipt.ID] = receipt

}

// GetReceipt gets a receipt by ID

func (s \*Store) GetReceipt(id string) (\*Receipt, bool) {

s.mu.Lock()

defer s.mu.Unlock()

receipt, found := s.receipts[id]

return receipt, found

}

**Step 3: Handlers for the API Endpoints**

**// handler.go**

package main

import (

"encoding/json"

"net/http"

"github.com/google/uuid"

)

// ProcessReceiptHandler handles the POST request to process a receipt

func ProcessReceiptHandler(store \*Store) http.HandlerFunc {

return func(w http.ResponseWriter, r \*http.Request) {

var receipt Receipt

if err := json.NewDecoder(r.Body).Decode(&receipt); err != nil {

http.Error(w, err.Error(), http.StatusBadRequest)

return

}

// Generate a unique ID for the receipt and calculate points

receipt.ID = uuid.New().String()

receipt.CalculatePoints()

// Save the receipt in the store

store.SaveReceipt(&receipt)

// Respond with the receipt ID

json.NewEncoder(w).Encode(map[string]string{"id": receipt.ID})

}

}

// GetPointsHandler handles the GET request to retrieve points for a receipt

func GetPointsHandler(store \*Store) http.HandlerFunc {

return func(w http.ResponseWriter, r \*http.Request) {

id := r.URL.Path[len("/receipts/"):]

receipt, found := store.GetReceipt(id)

if !found {

http.NotFound(w, r)

return

}

// Respond with the points awarded

json.NewEncoder(w).Encode(map[string]int{"points": receipt.PointsAwarded})

}

}

**Step 4: Main Application and Server Setup**

// main.go

package main

import (

"log"

"net/http"

)

func main() {

store := NewStore()

http.Handle("/receipts/process", ProcessReceiptHandler(store))

http.Handle("/receipts/", GetPointsHandler(store))

log.Println("Server starting on port 8080...")

if err := http.ListenAndServe(":8080", nil); err != nil {

log.Fatalf("Failed to start server: %v", err)

}

}

**Dockerization**

**# Dockerfile**

FROM golang:1.16

WORKDIR /app

COPY go.mod go.sum ./

RUN go mod download

COPY . .

RUN go build -o receipt-processor .

CMD ["./receipt-processor"]

And a **docker-compose.yml** file:

# docker-compose.yml

version: '3.8'

services:

receipt-processor:

build: .

ports:

- "8080:8080"

**Run on Local Host:**

- "8080:8080"