Введение

Вы почти закончили обучение и дошли до завершающей части курса, большая часть теории и практики осталась позади. Мы гордимся вашим трудолюбием и упорством! Теперь осталось совсем немного — выполнить финальную работу.

Зачем выполнять финальную работу? Это кейс в ваше портфолио, который можно показывать работодателю. Это полезно, даже если прямо сейчас вы не ищете работу: например, для роста внутри компании и для закрепления ваших знаний, полученных в курсе.

Поэтому давайте соберёмся и сделаем это. Будет интересно!

Тема вашего проекта

Сетевой многопоточный сервис для StatusPage

Перед вами подробное описание будущего проекта, который станет отличным дополнением вашего портфолио. Здесь есть всё, что вам нужно, чтобы справиться с поставленной задачей.

Содержание:

Описание задачи

Вы пришли работать разработчиком в компанию занимающуюся провайдингом современных средств коммуникации. Компания предоставляет инструменты и API для автоматизации работы систем SMS, MMS, Голосовых звонков и Email. География клиентов распространяется на 160 стран и компания быстро растёт. Требуется всё больше ресурсов со стороны службы поддержки и было принято решение снизить количество заявок с помощью создания страниц информирования клиентов о текущем состоянии систем. С помощью этих страниц компания планирует снизить количество однотипных вопросов и высвободить время агентов службы поддержки для решения более сложных задач.

В числе прочего были составлены страницы с ответами на часто задаваемые вопросы, уведомления о неполадках и истории инцидентов, чтобы клиенты могли самостоятельно проверять действующие системы на работоспособность. Поскольку компания работает на широкую аудиторию и распространена по всему миру, практически невозможно уследить за всеми изменениями вручную.

Поэтому каждое подразделение компании самостоятельно контролирует работу поставщиков услуг в автоматизированном режиме храня эти данные. Вашей задачей становится финализировать проект, объединив эти данные и разработав небольшой сетевой сервис, который будет принимать запросы по сети и возвращать данные о состоянии систем компании. Эти данные будут выводиться на web страницу сайта компании под названием StatusPage и содержать в себе географию и статусы сервисов. Так клиенты смогут проверить свой регион на наличие ошибок прежде чем обращаться в службу поддержки.

Ниже вы найдёте все технические подробности реализации сетевого демона, которые помогут вам создать работающее приложение. Они разбиты на несколько небольших этапов. По каждому этапу подробно расписано, что необходимо сделать и как проверить конечный результат.

Если у вас появятся вопросы, смело обращайтесь к вашему куратору.

Этап 1. Подготовка

Цель

Подготовить всё необходимое программное обеспечение, чтобы начать программировать сетевой сервис.

Что нужно сделать

1. Установите на свой компьютер среду разработки GoLand, если она ещё не установлена
2. Установите на свой компьютер git если он еще не установлен
3. Ознакомьтесь с кодами стран в формате ISO 3166-1 alpha-2, этот формат используется в проекте для идентификации стран
4. Ознакомьтесь с форматом файлов CSV, часть данных в проекте хранится в этом формате
5. Создайте директорию рядом с проектом финальной работы, перейдите в нее и склонируйте проект симулятора данных с github в отдельную директорию с помощью команды

```
git clone https://github.com/antondzhukov/skillbox-diploma ./
```

6. В соответствии с файлом Readme в проекте симулятора запустите проект. Проект сгенерирует нужные файлы и продолжит работать для обращения к нему по API для получения дополнительных данных которые мы рассмотрим ниже

На этапе сбора данных рекомендуется проверять работу функций сбора добавляя их в выполнение внутри функции main() и выводя результат в консоль с помощью функции fmt.Printf(). Или с помощью дебаггера.

Симулятор при каждом запуске генерирует новый набор данных. Часть из данных намеренно повреждается. Ваша цель не только получать данные, но и проверять их корректность в соответствии с заданием. Все некорректные данные должны быть проигнорированы в результирующих наборах.

Этап 2. Сбор данных о системе SMS

Цель

Реализовать функцию получения данных о состоянии системы SMS из файла формата CSV

Что нужно сделать

Напишите функцию, которая будет читать всё содержимое из файла, далее обходить содержимое построчно и разбирать строки на показатели

1. Для чтения из файла должен использоваться пакет ioutil
2. Файл записан в формате CSV
3. Данные в строках разделены с помощью знака ;. Чтобы разбить строку на поля можно использовать функцию strings.Split()
4. Каждая строка должна содержать 4 поля (alpha-2 код страны, пропускная способность канала от 0% до 100%, среднее время ответа в ms, название компании провайдера). Строки содержащие отличное количество полей не должны попадать в результат работы функции. Проверить количество элементов в срезе можно с помощью функции len()
5. Некоторые строки могут быть повреждены, их нужно пропускать и не записывать в результат выполнения функции
6. В результат допускаются только страны прошедшие проверку на существование по alpha-2 коду. Рекомендуется скопировать список стран себе в проект, составить по нему map и написать функцию проверяющую существование страны
7. В результат допускаются только корректные провайдеры. Допустимые провайдеры: Topolo, Rond, Kildy. Все некорректные провайдеры нужно пропускать и не добавлять в результат работы функции
8. Строки в которых меньше 4-х полей данных не допускаются
9. Результатом работы функции должен быть срез содержащий структуры из 4-х полей с набором данных каждой строки. Например, строка BG;42;501;Rond может быть трансформирована в структуру

```
type SMSData struct {
        Country  string
        Bandwidth string
        ResponseTime string
        Provider string
```

```
}
```

Как проверить работу функции

1. Запустите симулятор.
2. Симулятор сгенерирует файл sms.data в своей директории.
3. Запустите свой проект для вывода результата в консоль с помощью функции fmt.Printf() или для проверки результата работы функции дебаггером
4. Откройте файл sms.data в директории проекта симулятора
5. Сравните содержимое файла с результатом вывода консоли или содержимым переменной в отладчике

Пример файла

U5;41910;Topol
US;36;1576;Rond
GB28495Topolo
F2;9;484;Topolo
BL;68;1594;Kildy

В результирующий набор должны попасть 2 структуры содержащие данные из строк.

US;36;1576;Rond
BL;68;1594;Kildy

Строка 1 не попадает в результат из-за некорректности провайдера. Строка 3 не попадёт из отсутствия разделителей и возможности получить 4 элемента данных. Строка 4 не попадет из-за некорректного кода страны

Этап 3. Сбор данных о системе MMS

Цель

Реализовать функцию получения данных о состоянии системы MMS опрашивая API системы через HTTP GET запрос . Структура в целом похожа на действия с SMS, но возвращается в виде json

Что нужно сделать

Напишите функцию, которая будет отправлять GET запрос по адресу 127.0.0.1:8383 и разбирать полученный ответ в срез структур и возвращать его

1. Используя пакет https://golang.org/pkg/net/http/ отправьте GET запрос по адресу 127.0.0.1:8383
2. Полученный ответ переведите в срез []byte с помощью функции io.ReadAll
3. Полученный срез переведите в срез структур вида

```
type MMSData struct {
        Country  string          `json:"country"`
        Provider string          `json:"provider"`
        Bandwidth string         `json:"bandwidth"`
        ResponseTime string `json:"response_time"`
}
```

4. Обратите внимание, API не всегда возвращает корректный ответ, проверяйте код ответа, если он равен 200, можно переводить полученный ответ в структуры, если код равен 500 - произошла ошибка
5. Состав полей соответствует структуре из п.3. Для разбора полученного json в срез структур используйте функцию json.Unmarshal
6. Поле country может содержать только alpha-2 код из общего списка стран. Если страна отсутствует в общем списке, этот элемент нужно удалить из результата работы функции
7. Допустимые провайдеры MMS: Topolo, Rond, Kildy. Если провайдер элемента не соответствует одному из представленных, этот элемент должен быть удален из результатов работы функции
8. В случае ошибки со стороны API симулятора или разбора json в структуру возвращайте в виде ответа пустой срез []MMSData

Как проверить работу функции

6. Запустите симулятор.
7. Симулятор откроет соединение по адресу 127.0.0.1:8383.
8. Запустите свой проект для вывода результата в консоль с помощью функции fmt.Printf() или для проверки результата работы функции дебаггером
9. Выведите в консоль или дебаггер результат полученный из GET запроса.
10. В случае успеха вы должны увидеть строку такого вида, В то же время код ответа сервера размещенный в Response.StatusCode равен 200

```
{
```

```
  "data": [
    {
      "country": "EN",
      "provider": "Topolo",
      "bandwidth": "98",
      "response_time": "1920"
    },
    {
      "country": "RU",
      "provider": "Kildy",
      "bandwidth": "3",
      "response_time": "511"
    }
  ]
}
```

11. В случае ошибки вы должны увидеть пустую строку. В то же время код ответа сервера размещенный в Response.StatusCode равен 500
12. Конвертируйте полученный json в срез структур
13. Отфильтруйте его согласно условиям страны и провайдера
14. Полученный срез структур выведите в консоль или дебаггер.
15. Сравните полученный срез структур с ответом симулятора
16. Проверьте, что элементы корректно отфильтрованы

Этап 4. Сбор данных о системе Voice Call

Описание

Общий принцип работы VoiceCall похож на SMS, но в данном случае большее широкий набор значений. Компания старается улучшать качество системы, поэтому делает тестовые звонки, позволяет записывать разговоры и оценивает параметры влияющие на разговор

Цель

Реализовать функцию получения данных о состоянии системы VoiceCall из файла формата CSV. Разбор файла в целом похож на действия с SMS, но содержит дополнительные условия

Что нужно сделать

Напишите функцию, которая будет читать всё содержимое из файла, далее обходить содержимое построчно и разбирать строки на показатели

1. Для чтения из файла должен использоваться пакет ioutil
2. Файл записан в формате CSV
3. Данные в строках разделены с помощью знака ;. Чтобы разбить строку на поля можно использовать функцию strings.Split()
4. Каждая строка должна содержать 8 полей (alpha-2 код страны, текущая нагрузка в процентах, среднее время ответа, провайдер, стабильность соединения, TTFB, чистота связи, медиана длительности звонка). Строки содержащие отличное количество полей не должны попадать в результат работы функции. Проверить количество элементов в срезе можно с помощью функции len()
5. Некоторые строки могут быть повреждены, их нужно пропускать и не записывать в результат выполнения функции
6. В результат допускаются только страны прошедшие проверку на существование по alpha-2 коду.
7. В результат допускаются только корректные провайдеры. Допустимые провайдеры: TransparentCalls, E-Voice, JustPhone. Все некорректные провайдеры нужно пропускать и не добавлять в результат работы функции
8. Строки в которых меньше 8-х полей данных не допускаются
9. Все целочисленные данные должны быть приведены к типу int
10. Все числа с плавающей точкой должны быть приведены к типу float32
11. Результатом работы функции должен быть срез содержащий структуры из 4-х полей с набором данных каждой строки. Например, строка MC;25;1274;E-Voice;0.51;46;68;24 может быть трансформирована в структуру

```
type VoiceCallData struct {
 Country  string
 Bandwidth string
 ResponseTime string
 Provider string
 ConnectionStability float32
 TTFB int
 VoicePurity int
 MedianOfCallsTime int
}
```

Как проверить работу функции

17. Запустите симулятор.
18. Симулятор сгенерирует файл sms.data в своей директории.
19. Запустите свой проект для вывода результата в консоль с помощью функции fmt.Printf() или для проверки результата работы функции дебаггером
20. Откройте файл voice.data в директории проекта симулятора
21. Сравните содержимое файла с результатом вывода консоли или содержимым переменной в отладчике

Пример файла

```
BL;58;930;E-Voic;0.65;738;83;52
AT40673Transparentalls;0.62;581;38;10
BG;40;609;E-Voice;0.86;160;36;5
DK;11;743;JustPhone;0.67;82;74;41
```

В результирующий набор должны попасть 2 структуры содержащие данные из строк.

```
BG;40;609;E-Voice;0.86;160;36;5
DK;11;743;JustPhone;0.67;82;74;41
```

Строка 1 не попадает в результат из-за некорректности провайдера. Строка 2 не попадёт из отсутствия разделителей и возможности получить 4 элемента данных.

Этап 5. Сбор данных о системе Email

Описание

Для оценки качества доставки писем компания самостоятельно раз в минуту отправляет письма каждому провайдеру на почтовые ящики распределенные по всему миру и с помощью API проверяет через какое время приходит письмо. Письма отправляются от разных провайдеров чтобы получить медианное время доставки. Значение 0 в значении времени доставки означает что письмо не было получено в течение часа.

Цель

Реализовать функцию получения данных о состоянии системы Email из файла формата CSV. Разбор файла в целом похож на действия с SMS и VoiceCall, но содержит дополнительные условия

Что нужно сделать

Напишите функцию, которая будет читать всё содержимое из файла, далее обходить содержимое построчно и разбирать строки на показатели

1. Для чтения из файла должен использоваться пакет ioutil
2. Файл записан в формате CSV
3. Данные в строках разделены с помощью знака ;. Чтобы разбить строку на поля можно использовать функцию strings.Split()
4. Каждая строка должна содержать 3 поля (alpha-2 код страны, провайдер, среднее время доставки писем в ms). Строки содержащие отличное количество полей не должны попадать в результат работы функции. Проверить количество элементов в срезе можно с помощью функции len()
5. Некоторые строки могут быть повреждены, их нужно пропускать и не записывать в результат выполнения функции
6. В результат допускаются только страны прошедшие проверку на существование по alpha-2 коду.
7. В результат допускаются только корректные провайдеры. Допустимые провайдеры: "Gmail", "Yahoo", "Hotmail", "MSN", "Orange", "Comcast", "AOL", "Live", "RediffMail", "GMX", "Protonmail", "Yandex", "Mail.ru",
8. Все некорректные провайдеры нужно пропускать и не добавлять в результат работы функции
9. Строки в которых меньше 3-х полей данных не допускаются
10. Время доставки письма должно быть приведено к типу int
11. Результатом работы функции должен быть срез содержащий структуры из 3-х полей с набором данных каждой строки. Например, строка RU;Gmail;581 может быть трансформирована в структуру

```
type EmailData struct {
 Country  string
 Provider string
 DeliveryTime int
}
```

Как проверить работу функции

22. Запустите симулятор.
23. Симулятор сгенерирует файл sms.data в своей директории.
24. Запустите свой проект для вывода результата в консоль с помощью функции fmt.Println() или для проверки результата работы функции дебаггером
25. Откройте файл email.data в директории проекта симулятора
26. Сравните содержимое файла с результатом вывода консоли или содержимым переменной в отладчике

Пример файла

T;Gmail;511
AT;Yahoo274
AT;Hotmail;487

В результирующий набор должна попасть 1 структура содержащая данные из строки

AT;Hotmail;487

Строка 1 не попадает в результат из-за некорректности страны. Строка 2 не попадёт из отсутствия разделителей и возможности получить 3 элемента данных.

Этап 6. Сбор данных о системе Billing*

Описание

Помимо систем провайдинга услуг у компании есть автоматизированная система биллинга для реализации оплаты услуг в ручном и автоматизированном режиме. Команда биллинга контролирует свои сервисы как простое состояние работает / не работает и для экономии места использует битовую маску

Цель

Реализовать функцию получения данных о состоянии системы Billing из файла содержащего битовую маску состояния систем. Каждый бит отвечает за состояние отдельной системы. Системы по порядку: создание клиента, оплата, выплата, платежи по подписке, контроль мошенничества, страница оплаты

Что нужно сделать

1. Для чтения из файла должен использоваться пакет ioutil
2. Файл содержит битовую маску
3. Необходимо помнить что отсчет битов в битовой маске начинается справа
4. Полученная строка должна быть разбита на срез байтов и с помощью возведения в степень каждого бита интерпретирована в число. Сумма степеней каждого бита должна быть присвоена переменной с типом uint8.

   Например мы имеем маску вида 00010011

   Все нулевые биты мы пропускаем. Каждый бит с единицей возводится в степень своего номера. Таким образом можно посчитать число следующим способом (справа налево): 2^0 + 2^1 + 0 + 0 + 2^4 + 0 + 0 + 0 = 19

   Каждый бит со значением 1 был использован для возведения двойки в степень номера бита. Мы возвели в степень 0, 1, 4 биты и сложили результаты, остальные проигнорировали.

5. С помощью полученного числа и булевых операций нужен выделить каждый бит и проверить его на соответствие 1. Если бит проходит проверку значение поля в структуре должно быть true, если равен 0 то false

6. Результаты булевых операций нужно сохранить в структуру вида

```
type BillingData struct {
 CreateCustomer bool
 Purchase bool
 Payout bool
 Recurring bool
 FraudControl bool
 CheckoutPage bool
}
```

Как проверить работу функции

Перед работой с файлом лучше провести несколько ручных проверок предварительно посчитанных битовых масок. А уже после приступать к подсчету из файла. Можно подменять значения в файле для тестирования. В качестве проверяемых на этапе

тестирования можно использовать 3 маски:

00010011 = 19 (CreateCustomer, Purchase и FraudControl должны быть true, остальное false)
00000011 = 3 (CreateCustomer и Purchase должны быть true, остальное false)
00110011 = 51 (CreateCustomer, Purchase, FraudControl и CheckoutPage должны быть равны true, остальное false)

Проверьте полученный результат с помощью вывода структуры в консоль с помощью функции fmt.Printf()

Этап 7. Сбор данных о системе Support

Цель

Написать функцию, которая будет получать данные о текущей загрузке команды службы поддержки по API для дальнейшего прогноза потенциального времени ожидания ответа

Что нужно сделать

Напишите функцию, которая будет отправлять GET запрос по адресу 127.0.0.1:8484 и разбирать полученный ответ в срез структур и возвращать его

1. Используя пакет https://golang.org/pkg/net/http/ отправьте GET запрос по адресу 127.0.0.1:8484
2. Полученный ответ переведите в срез []byte с помощью функции io.ReadAll
3. Полученный срез переведите в срез структур вида

```
type SupportData struct {
	Topic string	`json:"topic"`
	ActiveTickets int `json:"active_tickets"`
}
```

4. Обратите внимание, API не всегда возвращает корректный ответ, проверяйте код ответа, если он равен 200, можно переводить полученный ответ в структуры, если код равен 500 - произошла ошибка
5. Состав полей соответствует структуре из п.3. Для разбора полученного json в срез структур используйте функцию json.Unmarshal
6. В случае ошибки со стороны API симулятора или разбора json в структуру возвращайте в виде ответа пустой срез []SupportData

Как проверить работу программы

1. Запустите симулятор.
2. Симулятор откроет соединение по адресу 127.0.0.1:8484.
3. Запустите свой проект для вывода результата в консоль с помощью функции fmt.Printf() или для проверки результата работы функции дебаггером
4. Выведите в консоль или дебаггер результат полученный из GET запроса.
5. В случае успеха вы должны увидеть строку такого вида, В то же время код ответа сервера размещенный в Response.StatusCode равен 200

```
{
  "data": [
    {
      "topic": "SMS",
      "active_tickets": 3
    },
    {
      "topic": "MMS",
      "active_tickets": 9
    },
    {
      "topic": "Billing",
      "active_tickets": 0
    }
  ]
}
```

6. В случае ошибки вы должны увидеть пустую строку. В то же время код ответа сервера размещенный в Response.StatusCode равен 500
7. Конвертируйте полученный json в срез структур
8. Полученный срез структур выведите в консоль или дебаггер.
9. Сравните полученный срез структур с ответом симулятора

Этап 8. Сбор данных о системе истории инцидентов

Описание

В общем потоке данных система истории инцидентов просто содержит массив инцидентов за последнюю неделю. Мы будем возвращать нашим пользователям историю инцидентов, чтобы разгрузить саппорт от однообразных вопросов. Например, у нас происходят неполадки в системе SMS, саппорт создает инцидент. Пользователи на нашей странице StatusPage могут посмотреть что у нас неполадки, и на какое-то время отключить свои системы до завершения работ по восстановлению не создавать новых тикетов саппорту. Таким образом мы сообщаем нашим пользователям о возможных и бывших проблемах с сервисами.

Цель

Напишите функцию, которая будет отправлять GET запрос по адресу 127.0.0.1:8585 и разбирать полученный ответ в срез структур и возвращать его

7. Используя пакет https://golang.org/pkg/net/http/ отправьте GET запрос по адресу 127.0.0.1:8585
8. Полученный ответ переведите в срез []byte с помощью функции io.ReadAll
9. Полученный срез переведите в срез структур вида

```
type IncidentData struct {
	Topic  string	`json:"topic"`
	Status  string		`json:"status"` // возможные статусы active и closed
}
```

10. Обратите внимание, API не всегда возвращает корректный ответ, проверяйте код ответа, если он равен 200, можно переводить полученный ответ в структуры, если код равен 500 - произошла ошибка
11. Состав полей соответствует структуре из п.3. Для разбора полученного json в срез структур используйте функцию json.Unmarshal
12. В случае ошибки со стороны API симулятора или разбора json в структуру возвращайте в виде ответа пустой срез []IncidentData

Как проверить работу программы

10. Запустите симулятор.
11. Симулятор откроет соединение по адресу 127.0.0.1:8585.

12. Запустите свой проект для вывода результата в консоль с помощью функции fmt.Printf() или для проверки результата работы функции дебаггером
13. Выведите в консоль или дебаггер результат полученный из GET запроса.
14. В случае успеха вы должны увидеть строку такого вида, В то же время код ответа сервера размещенный в Response.StatusCode равен 200

```
{
  "data": [
    {
      "topic":   "Billing isn’t allowed in US",
      "status": “closed”
    },
    {
      "topic":   "Wrong SMS delivery time",
      "status": “active”
    },
    {
      "topic": "Support overloaded because of EU affect",
      "active_tickets": “active”
    }
  ]
}
```

15. В случае ошибки вы должны увидеть пустую строку. В то же время код ответа сервера размещенный в Response.StatusCode равен 500
16. Конвертируйте полученный json в срез структур
17. Полученный срез структур выведите в консоль или дебаггер.

Сравните полученный срез структур с ответом симулятора

Этап 9. Подготовка сетевого сервиса

Цель

Подготовить программу к работе по сети. Организовать открытие сетевого соединения и провести первичное тестирование обработки запросов

Что нужно сделать

1. С помощью пакетов net/http и https://github.com/gorilla/mux создайте соединение с локальным адресом и портом (например localhost:8282). Используйте функцию http.Server()
2. С помощью функции mux.NewRouter() создайте новый роутер из добавьте ему обработку адреса “/” на функцию handleConnection (используйте документацию на странице https://github.com/gorilla/mux для примеров использования роутера)
3. Напишите функцию handleConnection по примеру функции ArticlesCategoryHandler из документации https://github.com/gorilla/mux. Ваша функция должна возвращать только слово “ok”
4. Назначьте серверу из п.1 получившийся роутер
5. Откройте соединение функцией ListenAndServe() сервера

Как проверить работу программы

1. Запустите программу
2. Откройте любой браузер и введите сетевой адрес, который вы указали при создании соединения (например http://localhost:8282/)
3. Нажмите Enter, на странице должно появится слово “ok”

Этап 10. Подготовка структур.

Цель

Перед тем как возвращать данные по запросу, нужно подготовить структуры к автоматической конвертации в json

Что нужно сделать

1. Каждая структура содержит свойства, напротив которого нужно проставить тег `json:"название_поля"`. Название поля должно быть траснформировано в Snake case. Например свойство “Country” будет трансформировано в “country”, а ProviderName в provider_name.
2. Дополните все свойства структур полученных на этапе сбора данных тегами json с трансформированным названием поля
3. Подготовьте родительские структуры ResultT и ResultSetT, которые будут содержать следующие поля

```
type ResultT struct {
  Status bool              `json:"status"` // true, если все этапы сбора данных прошли успешно, false во всех остальных случаях
```

```
 Data      ResultSetT `json:"data"` // заполнен, если все этапы сбора данных прошли успешно, nil во всех остальных случаях
 Error     string          `json:"error"` // пустая строка если все этапы сбора данных прошли успешно, в случае ошибки заполнено текстом ошибки (детали ниже)
}
```

```
type ResultSetT struct {
 SMS         [][]SMSData        `json:"sms"`
 MMS         [][]MMSData        `json:"mms"`
 VoiceCall   []VoiceCallData    `json:"voice_call"`
 Email       map[string][][]EmailData  `json:”email”`
 Billing     BillingData        `json:”billing”`
 Support     []int              `json:”support”`
 Incidents   []IncidentData     `json:”incident”`
}
```

Как проверить работу программы

1. Напишите функцию-пустышку, в которой ваша программа будет создавать родительские структуры
2. Конвертируйте структуры с помощью функции json.Marshal в формат json
3. Выведите результат в консоль
4. Вы должны увидеть json соответствующий тегам json в структурах

Этап 11. Сортировка и фильтрация данных

Описание

Полученные наборы данных необходимо подготовить для возврата браузеру в нужном формате и количестве. Для отображения данные должны быть отсортированы и дополнительно отфильтрованы чтобы показывать их клиентам в удобном для просмотра виде.

Цель

Обойти все этапы сбора данных и составить результирующий набор с помощью родительских структур ResultT и ResultSetT

Что нужно сделать

1. Напишите функцию getResultData(): ResultSetT которая будет обходить все этапы сбора данных и возвращать готовый для возврата набор
2. Получите данные по sms и подготовьте 2 отсортированных списка (используйте приёмы из модуля “Сортировка массивов”). Подмените коды стран в структурах SMSData на полные названия стран основываясь на списки alpha-2 кодов. Первый набор должен быть отсортирован по названию по названию провайдера от A до Z. Второй набор должен быть отсортирован по названию страны от A до Z. Объедините оба набора в срез. У вас должен получиться срез вида [][]SMSData содержащий 2 среза с отсортированными данными. Запишите данные в свойство SMS структуры ResultSetT
3. Получите данные по mms и подготовьте 2 отсортированных списка (используйте приёмы из модуля “Сортировка массивов”). Подмените коды стран в структурах MMSData на полные названия стран основываясь на списки alpha-2 кодов. Первый набор должен быть отсортирован по названию по названию провайдера от A до Z. Второй набор должен быть отсортирован по названию страны от A до Z. Объедините оба набора в срез. У вас должен получиться срез вида [][]MMSData содержащий 2 среза с отсортированными данными. Запишите данные в свойство MMS структуры ResultSetT
4. Получите данные по voice call. Эти данные никак не нужно модифицировать. Можно напрямую заполнить []VoiceCall в ResultSetT
5. Получите данные по Email. Вам нужно отсортировать всех провайдеров в каждой стране по показателю среднего времени доставки письма и составить из этого карту с ключем кода страны (map[string][]EmailData) и 2-мя срезами внутри. Первый должен содержать 3 самых быстрых провайдера, второй 3 самых медленных. Запишите данные в свойство Email структуры ResultSetT
6. *Сбор данных о системе Billing. Эти данные никак не нужно модифицировать. Можно напрямую заполнить Billing в ResultSetT. Если вы не выполняли это задание, пропустите заполнение свойства Billing
7. Получите данные о системе Support. Средняя пропускная способность саппорта 18 тикетов в час. До 9 - саппорт не нагружен, 9 - 16 средненагружен, > 16 перегружен. Саппорт разделен на специалистов по темам, если в какой-то теме мало запросов их перебрасывают в другие. При этом профильные задачи решаются в первую очередь, поэтому общая нагрузка не влияет на решение профильных задач. Нужно возвращать наружу общее состояние нагрузки в виде инта (1, 2, 3) по степени нагрузки. Нужно возвращать потенциальное время ожидания по каждому профилю задач. Всего в саппорте 7 специалистов. Мы используем упрощенную система расчета. (60 минут / 18 тикетов/час) = количество минут, которое 7 человек в среднем тратят на 1 тикет. Исходя из среднего количества минут на тикет и суммы открытых тикетов посчитайте потенциальное время ожидания ответа на новый тикет просто умножив количество открытых тикетов на количество минут на тикет. Запишите в ResultSetT срез из 2-х int, первый из которых будет показывать загруженность саппорта (1-3), а второй среднее время ожидания ответа
8. Получите данные об истории инцидентов. Отсортируйте полученные данные так, чтобы все инциденты со статусом active оказались наверху списка, а

остальные ниже. Порядок остальной сортировки не важен. Запишите данные в свойство Support структуры ResultSetT

Как проверить работу программы

1. Запустите программу и функцию getResultData()
2. Выведите результат работы функции в консоль
3. Сверьте сортировку и фильтрацию данных с содержимым полученным из симулятора

Этап 12. API состояния систем и ответы в случае ошибок

Цель

Что нужно сделать

1. Завершите написание функции handleConnection
2. Создайте переменную и запишите в неё структуру ResultT
3. Выполните функцию getResultData и запишите её результат в переменную
4. Проверьте что все поля ResultSetT заполнены (если вы не выполняли сбор данных по Billing его проверять не нужно)
5. Если все поля заполнены, установите свойство Status структуры ResultT в true, если есть пропуски установите его в false
6. Если есть пропуски в поле ResultT.Error в виде ошибки запишите “Error on collect data”
7. Если сбор данных прошле без ошибок заполните ResultT.Data структурой ResultSetT. Если ошибки были не заполняйте это свойство.
8. С помощью функции json.Marshal запишите в новую переменную содержимое полученной структуры ResultT в виде json
9. Замените слово “ok” в выводе на переменную содержащую результирующий json

Как проверить работу программы

1. Запустите программу
2. Откройте любой браузер и введите сетевой адрес, который вы указали при создании соединения (например http://localhost:8282/)
3. Нажмите Enter, на странице должен появиться json содержащий результат работы программы с отфильтрованными и отсортированными данными
4. В функции handleConnection после сбора данных намеренно присвойте свойству ResultSetT.SMSData nil
5. Обновите страницу в браузере
6. На экране должен быть json с ошибкой

Дополнительные задачи

Если выполнение основного технического задания вам покажется простым, предлагаем вам дополнительные задачи. Вы можете выбирать любые из них и самостоятельно реализовать, сообщив об этом своему куратору:

1. Добейтесь большей асинхронности, пусть каждый сбор данных выполняется в отдельном потоке с помощью ключевого слова go. Для этого можно использовать каналы. Не забудьте про использование mutex при конкурентном доступе к одним и тем же глобальным переменным
2. Улучшите качество кода. Напишите функции проверки существования кода страны, провайдеров и других общий операций с данными
3. Есть более дешевые способы проверки целостности CSV строки. Попробуйте придумать как определить количество колонок в CSV без разложения на срез и использования функции strings.Split
4. Выдавайте отсортированные и отфильтрованные данные сразу из функций сбора. В этом случае нужно будет лишь заполнить нужные свойства результирующей структуры
5. Состояние систем в реальных проектах не меняется каждую долю секунды. Для улучшения быстродействия можно хранить полученный результат в течение 30-ти секунд в глобальной переменной и возвращать его без опроса систем на каждый запрос к вашему сервису. Для очистки результата можно использовать тикер

Рекомендации

1. Избегать повторов кода.
2. Именовать переменные, методы и классы в соответствии с правилами именования в Go, а также таким образом, чтобы их имена отражали их назначение.
3. Всегда выполняйте go fmt перед отправкой проекта на проверку
4. По возможности, не писать в коде методов комментариев. По коду должно быть понятно, что он делает.
5. По возможности, используйте алгоритмы сортировки с меньшей алгоритмической сложностью из уроков или другие, если сочтете их более оптимальными
6. Все повторяющиеся операции старайтесь выносить в отдельные функции. Это даёт переиспользование и меньшее редактирование кода при изменениях
7. Используйте дебаггер для отладки и тестирования

8. Создавайте общие структуры в глобальной области видимости для переиспользования в разных частях проекта