Детали проекта big_go как первый действующий макет

Общее описание см. README.md

Dockerfile для каждого сервиса

- **1. Dockerfile для генератора данных**

```
FROM golang:1.19-alpine

WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .

RUN go build -o /generator ./cmd/generator/main.go

EXPOSE 8080

CMD ["/generator"]
```

- **2. Dockerfile для коллектора**

```
FROM golang:1.19-alpine

WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .

RUN go build -o /collector ./cmd/collector/main.go

EXPOSE 8081

CMD ["/collector"]
```

- **3. Dockerfile для User1**

```
FROM golang:1.19-alpine
```

```
WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .

RUN go build -o /user1 ./cmd/user1/main.go

EXPOSE 8082

CMD ["/user1"]
```

- **4. Dockerfile для User2**

```
FROM golang:1.19-alpine

WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .

RUN go build -o /user2 ./cmd/user2/main.go

EXPOSE 8083

CMD ["/user2"]
```

Docker Compose файл - (docker-compose.yml)**

```
version: '3.8'

services:
  postgres:
    image: postgres:14
    container_name: big_go_postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: big_go
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - big_go_network
    healthcheck:
```

```
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5

  redis:
    image: redis:7
    container_name: big_go_redis
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    networks:
      - big_go_network
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 5s
      timeout: 5s
      retries: 5

  rabbitmq:
    image: rabbitmq:3-management
    container_name: big_go_rabbitmq
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      RABBITMQ_DEFAULT_USER: guest
      RABBITMQ_DEFAULT_PASS: guest
    volumes:
      - rabbitmq_data:/var/lib/rabbitmq
    networks:
      - big_go_network
    healthcheck:
      test: ["CMD", "rabbitmqctl", "status"]
      interval: 10s
      timeout: 5s
      retries: 5

  generator:
    build:
      context: .
      dockerfile: docker/generator/Dockerfile
    container_name: big_go_generator
    depends_on:
      rabbitmq:
        condition: service_healthy
    networks:
      - big_go_network
    volumes:
      - ./config_go.json:/app/config_go.json
      - ./config_rabbitmq.json:/app/config_rabbitmq.json
    environment:
      - RABBITMQ_HOST=rabbitmq
```

```
      - RABBITMQ_PORT=5672
      - RABBITMQ_USER=guest
      - RABBITMQ_PASSWORD=guest


  collector:
    build:
      context: .
      dockerfile: docker/collector/Dockerfile
    container_name: big_go_collector
    depends_on:
      rabbitmq:
        condition: service_healthy
      postgres:
        condition: service_healthy
      redis:
        condition: service_healthy
    networks:
      - big_go_network
    volumes:
      - ./config_go.json:/app/config_go.json
      - ./config_rabbitmq.json:/app/config_rabbitmq.json
      - ./config_postgresql.json:/app/config_postgresql.json
      - ./config_redis.json:/app/config_redis.json
    environment:
      - RABBITMQ_HOST=rabbitmq
      - RABBITMQ_PORT=5672
      - RABBITMQ_USER=guest
      - RABBITMQ_PASSWORD=guest
      - POSTGRES_HOST=postgres
      - POSTGRES_PORT=5432
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=big_go
      - REDIS_HOST=redis
      - REDIS_PORT=6379


  user1:
    build:
      context: .
      dockerfile: docker/user1/Dockerfile
    container_name: big_go_user1
    depends_on:
      - collector
    ports:
      - "8082:8082"
    networks:
      - big_go_network
    environment:
      - COLLECTOR_HOST=collector
      - COLLECTOR_PORT=8081

  user2:
```

```
    build:
      context: .
      dockerfile: docker/user2/Dockerfile
    container_name: big_go_user2
    depends_on:
      - collector
    ports:
      - "8083:8083"
    networks:
      - big_go_network
    environment:
      - COLLECTOR_HOST=collector
      - COLLECTOR_PORT=8081

networks:
  big_go_network:
    driver: bridge

volumes:
  postgres_data:
  redis_data:
  rabbitmq_data:
```

Реализация основных компонентов

1. Модель данных (internal/models/models.go)

```
package models

import "time"

// SensorData представляет данные от датчиков
type SensorData struct {
    Meta MetaData `json:"meta"`
    Data DataPoint `json:"data"`
}

// MetaData содержит метаданные сообщения
type MetaData struct {
    Recipient  string    `json:"recipient"`  // User1 или User2
    PostID     int       `json:"post_id"`    // Номер поста (1-10)
    Address    int       `json:"address"`    // Адрес
    Timestamp  time.Time `json:"timestamp"`  // Временная метка
}

// DataPoint содержит данные измерений
type DataPoint struct {
    Temperature float64 `json:"temperature"` // Температура в градусах
Цельсия
    Pressure    float64 `json:"pressure"`    // Давление в мм.рт.ст.
```

```
    Humidity    float64 `json:"humidity"`    // Влажность в %
}
```

2. Генератор данных (cmd/generator/main.go)

```
package main

import (
    "big_go/config"
    "big_go/internal/services/generator"
    "encoding/json"
    "fmt"
    "log"
    "math/rand"
    "time"

    "github.com/streadway/amqp"
)

func main() {

    // Инициализация конфигурации RabbitMQ
    rabbitConfig, err := config.LoadRabbitMQConfig("config_rabbitmq.json")
    if err != nil {
        log.Fatalf("Ошибка загрузки конфигурации RabbitMQ: %v", err)
    }

    // Подключение к RabbitMQ
    conn, err := amqp.Dial(
        "amqp://" + rabbitConfig.User + ":" + rabbitConfig.Password +
            "@" + rabbitConfig.Host + ":" + fmt.Sprintf("%d",
rabbitConfig.Port) + "/" + rabbitConfig.VHost,
    )
    if err != nil {
        log.Fatalf("Ошибка подключения к RabbitMQ: %v", err)
    }
    defer conn.Close()

    // Создание канала
    ch, err := conn.Channel()
    if err != nil {
        log.Fatalf("Ошибка создания канала: %v", err)
    }
    defer ch.Close()

    // Объявление очереди
    q, err := ch.QueueDeclare(
        "sensor_data", // имя очереди
        true,          // durable
        false,         // delete when unused
        false,         // exclusive
```

```
        false,          // no-wait
        nil,            // arguments
    )
    if err != nil {
        log.Fatalf("Ошибка объявления очереди: %v", err)
    }

    // Инициализация генератора данных
    gen := generator.NewGenerator()

    // Запуск генерации данных
    for {
        // Генерация случайного интервала от 1 до 5 секунд
        interval := time.Duration(rand.Intn(4)+1) * time.Second

        // Генерация данных
        data := gen.GenerateData()

        // Сериализация данных в JSON
        jsonData, err := json.Marshal(data)
        if err != nil {
            log.Printf("Ошибка сериализации данных: %v", err)
            continue
        }

        // Публикация сообщения
        err = ch.Publish(
            "",     // exchange
            q.Name, // routing key
            false,  // mandatory
            false,  // immediate
            amqp.Publishing{
                ContentType: "application/json",
                Body:        jsonData,
            })
        if err != nil {
            log.Printf("Ошибка публикации сообщения: %v", err)
        } else {
            log.Printf("Отправлено сообщение: %s", string(jsonData))
        }

        // Ожидание перед следующей генерацией
        time.Sleep(interval)
    }
}
```

3. Сервис генератора (internal/services/generator/generator.go)

```
package generator

import (
```

```
    "big_go/internal/models"
    "math/rand"
    "time"
)

const post_numbers = 4
const address_numbers = 10
const recipient_numbers = 2

// Generator представляет генератор данных
type Generator struct {
    rand *rand.Rand
}

// NewGenerator создает новый экземпляр генератора
func NewGenerator() *Generator {
    return &Generator{
        rand: rand.New(rand.NewSource(time.Now().UnixNano())),
    }
}

// GenerateData генерирует случайные данные датчиков
func (g *Generator) GenerateData() models.SensorData {
    // Определяем получателя (User1 или User2)
    recipient := "User1"
    if g.rand.Intn(recipient_numbers) == 1 {
        recipient = "User2"
    }

    // Генерируем случайный номер поста (1-10)
    postID := g.rand.Intn(post_numbers) + 1

    // Генерируем случайный адрес
    address := g.rand.Intn(address_numbers) + 1

    // Создаем метаданные
    meta := models.MetaData{
        Recipient: recipient,
        PostID:    postID,
        Address:   address,
        Timestamp: time.Now(),
    }

    // Генерируем данные измерений
    data := models.DataPoint{
        Temperature: 22.0 + g.rand.Float64()*3.0,   // от 20 до +25
градусов
        Pressure:    740.0 + g.rand.Float64()*40.0, // от 740 до 780
мм.рт.ст.
        Humidity:    40.0 + g.rand.Float64()*40.0,  // от 40 до 80%
    }

    return models.SensorData{
        Meta: meta,
```

```
        Data: data,
    }
}
```

4. Коллектор (cmd/collector/main.go)

```
package main

import (
    "big_go/config"
    "big_go/internal/models"
    "bytes"
    "encoding/json"
    "fmt"
    "log"
    "net/http"

    "github.com/streadway/amqp"
)

func main() {
    // Инициализация конфигурации RabbitMQ
    rabbitConfig, err := config.LoadRabbitMQConfig("config_rabbitmq.json")
    if err != nil {
        log.Fatalf("Ошибка загрузки конфигурации RabbitMQ: %v", err)
    }

    // Подключение к RabbitMQ
    conn, err := amqp.Dial(
        "amqp://" + rabbitConfig.User + ":" + rabbitConfig.Password +
            "@" + rabbitConfig.Host + ":" + fmt.Sprintf("%d",
rabbitConfig.Port) + "/" + rabbitConfig.VHost,
    )
    if err != nil {
        log.Fatalf("Ошибка подключения к RabbitMQ: %v", err)
    }
    defer conn.Close()

    // Создание канала
    ch, err := conn.Channel()
    if err != nil {
        log.Fatalf("Ошибка создания канала: %v", err)
    }
    defer ch.Close()

    // Объявление очереди
    q, err := ch.QueueDeclare(
        "sensor_data", // имя очереди
        true,          // durable
        false,         // delete when unused
        false,         // exclusive
```

```
        false,          // no-wait
        nil,            // arguments
    )
    if err != nil {
        log.Fatalf("Ошибка объявления очереди: %v", err)
    }

    // Настройка потребителя сообщений
    msgs, err := ch.Consume(
        q.Name, // queue
        "",     // consumer
        true,   // auto-ack
        false,  // exclusive
        false,  // no-local
        false,  // no-wait
        nil,    // args
    )
    if err != nil {
        log.Fatalf("Ошибка регистрации потребителя: %v", err)
    }

    log.Println("Коллектор запущен. Ожидание сообщений...")

    // Обработка сообщений
    for d := range msgs {
        var sensorData models.SensorData
        err := json.Unmarshal(d.Body, &sensorData)
        if err != nil {
            log.Printf("Ошибка десериализации сообщения: %v", err)
            continue
        }

        log.Printf("Получено сообщение: %+v", sensorData)

        // Определяем, куда отправить данные
        var endpoint string
        if sensorData.Meta.Recipient == "User1" {
            endpoint = "http://user1:8082/data"
        } else {
            endpoint = "http://user2:8083/data"
        }

        // Отправляем данные соответствующему пользователю
        jsonData, err := json.Marshal(sensorData)
        if err != nil {
            log.Printf("Ошибка сериализации данных: %v", err)
            continue
        }

        resp, err := http.Post(endpoint, "application/json",
bytes.NewBuffer(jsonData))
        if err != nil {
            log.Printf("Ошибка отправки данных пользователю: %v", err)
            continue
```

```
        }
        resp.Body.Close()

        log.Printf("Данные успешно отправлены на %s", endpoint)
    }
}
```

5. Сервис коллектора (internal/services/collector/collector.go)

```
package collector

import (
    "big_go/internal/models"
    "bytes"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
)

// Collector представляет сервис коллектора данных
type Collector struct {
    user1Data  chan models.SensorData
    user2Data  chan models.SensorData
    httpClient *http.Client
}

// NewCollector создает новый экземпляр коллектора
func NewCollector() *Collector {
    c := &Collector{
        user1Data:  make(chan models.SensorData, 100),
        user2Data:  make(chan models.SensorData, 100),
        httpClient: &http.Client{},
    }

    // Запуск горутин для отправки данных пользователям
    go c.sendDataToUser("user1", c.user1Data)
    go c.sendDataToUser("user2", c.user2Data)

    return c
}

// ProcessData обрабатывает полученные данные и направляет их
соответствующему пользователю
func (c *Collector) ProcessData(data models.SensorData) error {
    log.Printf("Обработка данных для %s от поста %d", data.Meta.Recipient,
data.Meta.PostID)

    // Направление данных соответствующему пользователю
    switch data.Meta.Recipient {
    case "User1":
```

```
            c.user1Data <- data
        case "User2":
            c.user2Data <- data
        default:
            return fmt.Errorf("неизвестный получатель: %s",
data.Meta.Recipient)
        }

        return nil
}

// sendDataToUser отправляет данные соответствующему пользовательскому
сервису
func (c *Collector) sendDataToUser(userService string, dataChan <-chan
models.SensorData) {
        var endpoint string
        if userService == "user1" {
            endpoint = "http://user1:8082/data"
        } else {
            endpoint = "http://user2:8083/data"
        }

        for data := range dataChan {
            jsonData, err := json.Marshal(data)
            if err != nil {
                log.Printf("Ошибка сериализации данных для %s: %v",
userService, err)
                continue
            }

            resp, err := http.Post(endpoint, "application/json",
bytes.NewBuffer(jsonData))
            if err != nil {
                log.Printf("Ошибка отправки данных для %s: %v", userService,
err)
                continue
            }
            resp.Body.Close()

            log.Printf("Данные успешно отправлены %s", userService)
        }
}
```

6. Пользовательский сервис (cmd/user1/main.go)

```
package main

import (
    "big_go/internal/models"
    "log"
    "net/http"
```

```
    "time"

    "github.com/gin-gonic/gin"
)

func main() {
    r := gin.Default()

    // Канал для хранения последних полученных данных
    var latestData []models.SensorData

    // Обработчик для получения данных от коллектора
    r.POST("/data", func(c *gin.Context) {
        var data models.SensorData
        if err := c.ShouldBindJSON(&data); err != nil {
            c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
            return
        }

        log.Printf("User1 получил данные: %+v", data)
        // Подробное логирование полученных данных
        log.Printf("User1 получил данные:")
        log.Printf("  Метаданные:")
        log.Printf("    Получатель: %s", data.Meta.Recipient)
        log.Printf("    ID поста: %d", data.Meta.PostID)
        log.Printf("    Адрес: %d", data.Meta.Address)
        log.Printf("    Временная метка: %s",
data.Meta.Timestamp.Format(time.RFC3339))
        log.Printf("  Данные измерений:")
        log.Printf("    Температура: %.2f °C", data.Data.Temperature)
        log.Printf("    Давление: %.2f мм.рт.ст.", data.Data.Pressure)
        log.Printf("    Влажность: %.2f %%", data.Data.Humidity)

        // Добавление данных в список последних данных (максимум 100
записей)
        latestData = append(latestData, data)
        if len(latestData) > 100 {
            latestData = latestData[1:]
        }

        c.JSON(http.StatusOK, gin.H{"status": "success"})
    })

    // Обработчик для отображения последних данных
    r.GET("/", func(c *gin.Context) {
        c.HTML(http.StatusOK, "index.html", gin.H{
            "title": "User1 Dashboard",
            "data":  latestData,
        })
    })

    // Загрузка HTML шаблонов
    r.LoadHTMLGlob("internal/templates/*.html")
```

```
    // Запуск сервера
    log.Println("User1 сервис запущен на порту 8082")
    if err := r.Run(":8082"); err != nil {
        log.Fatalf("Ошибка запуска сервера: %v", err)
    }
}
```

7. Пользовательский сервис (cmd/user2/main.go)

```
package main

import (
    "big_go/internal/models"
    "log"
    "net/http"
    "time"

    "github.com/gin-gonic/gin"
)

func main() {
    r := gin.Default()

    // Канал для хранения последних полученных данных
    var latestData []models.SensorData

    // Обработчик для получения данных от коллектора
    r.POST("/data", func(c *gin.Context) {
        var data models.SensorData
        if err := c.ShouldBindJSON(&data); err != nil {
            c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
            return
        }

        log.Printf("User2 получил данные: %+v", data)
        // Подробное логирование полученных данных
        log.Printf("User1 получил данные:")
        log.Printf("  Метаданные:")
        log.Printf("    Получатель: %s", data.Meta.Recipient)
        log.Printf("    ID поста: %d", data.Meta.PostID)
        log.Printf("    Адрес: %d", data.Meta.Address)
        log.Printf("    Временная метка: %s",
data.Meta.Timestamp.Format(time.RFC3339))
        log.Printf("  Данные измерений:")
        log.Printf("    Температура: %.2f °C", data.Data.Temperature)
        log.Printf("    Давление: %.2f мм.рт.ст.", data.Data.Pressure)
        log.Printf("    Влажность: %.2f %%", data.Data.Humidity)

        // Добавление данных в список последних данных (максимум 100
записей)
        latestData = append(latestData, data)
```

```
        if len(latestData) > 100 {
            latestData = latestData[1:]
        }

        c.JSON(http.StatusOK, gin.H{"status": "success"})
    })

    // Обработчик для отображения последних данных
    r.GET("/", func(c *gin.Context) {
        c.HTML(http.StatusOK, "index.html", gin.H{
            "title": "User2 Dashboard",
            "data":  latestData,
        })
    })

    // Загрузка HTML шаблонов
    r.LoadHTMLGlob("internal/templates/*.html")

    // Запуск сервера
    log.Println("User2 сервис запущен на порту 8083")
    if err := r.Run(":8083"); err != nil {
        log.Fatalf("Ошибка запуска сервера: %v", err)
    }
}
```

8. HTML шаблон для отображения данных (internal/templates/index.html)

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ .title }}</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        h1 {
            color: #333;
        }
        table {
            width: 100%;
            border-collapse: collapse;
            margin-top: 20px;
        }
        th, td {
            border: 1px solid #ddd;
            padding: 8px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
```

```
        }
        tr:nth-child(even) {
            background-color: #f9f9f9;
        }
    </style>
</head>
<body>
    <h1>{{ .title }}</h1>

    <h2>Последние полученные данные</h2>

    <table>
        <tr>
            <th>Время</th>
            <th>Пост ID</th>
            <th>Адрес</th>
            <th>Температура (°C)</th>
            <th>Давление (мм.рт.ст.)</th>
            <th>Влажность (%)</th>
        </tr>
        {{ range .data }}
        <tr>
            <td>{{ .Meta.Timestamp }}</td>
            <td>{{ .Meta.PostID }}</td>
            <td>{{ .Meta.Address }}</td>
            <td>{{ .Data.Temperature }}</td>
            <td>{{ .Data.Pressure }}</td>
            <td>{{ .Data.Humidity }}</td>
        </tr>
        {{ end }}
    </table>

    <script>
        // Автоматическое обновление страницы каждые 5 секунд
        setTimeout(function() {
            location.reload();
        }, 5000);
    </script>
</body>
</html>
```