|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУ-КФ «Информатика и управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУ4-КФ «Программное обеспечение ЭВМ, информационные технологии»***

**ДОМАШНЕЕ ЗАДАНИЕ**

**«**Тайловый веб-сервер**»**

**ДИСЦИПЛИНА: «**Кроссплатформенная разработка ПО**»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИТД.Б-62 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Фарутина П.В.)  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Жуков И.О.)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга , 2019

### Цель:

Объединение всех наработок и полученных навыков в единый проект с четкой бизнес-целью.

### Задача:

Создать кеширующий тайловый веб-сервер для визуализации заранее подготовленных геоданных.

### Требования к сервису:

1. Использование только реальных геоданных
2. Набор геометрии и стилей фиксируется на старте проекта
3. Наличие API для получения индивидуальных тайлов
4. Живой рендер в формат PNG, кеш в памяти

### Уровни сложности:

Минимальный - простой сервер, кеш, наличие слоев фиксировано

Легкий - страница с отображением тайлов в виде карты

Средний - управление наличием слоев

Сложный - метатайлы, постоянный кеш (диск/бд)

Ужасный - то же, что и сложный, но выдержать нагрузку более 1000 запросов в секунду

### Решение:

Результатом выполнения программы является приложение с кеширующим сервером визуализирующее полученную карту РФ из прошлых лабораторных работ с возможностью масштабирования на карте мира.

Написанный сервер обрабатывает запросы с клиента и отправляет готовые изображения в виде массива байт. Для хранения отрендеренных изображений используется переменная cache, которая является ассоциативным массивом, ключ к значениям которого является сама строка запроса в виде tile/z/x/y.

Для работы с тайлами на стороне клиента используется библиотека leafletjs.

Используемые библиотеки:

"github.com/davvo/mercator"

"github.com/fogleman/gg"

geojson "github.com/paulmach/go.geojson"

### Исходный код программы:

package main

import (

"bytes"

"fmt" // пакет для форматированного ввода вывода

"html/template" // пакет для логирования

"image"

"image/png"

"io"

"math"

"math/rand"

"net/http" // пакет для поддержки HTTP протокола

"os"

"strconv"

"strings"

"github.com/davvo/mercator"

"github.com/fogleman/gg"

geojson "github.com/paulmach/go.geojson"

// пакет для работы с UTF-8 строками

)

const width, height = 512, 512

var cache map[string][]byte

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

t, err := template.ParseFiles("./index.html")

if err != nil {

fmt.Fprintf(w, err.Error())

}

t.ExecuteTemplate(w, "index", "hello")

}

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

key := r.URL.String()

keys := strings.Split(key, "/")

z, \_ := strconv.ParseFloat(keys[0], 64)

x, \_ := strconv.ParseFloat(keys[1], 64)

y, \_ := strconv.ParseFloat(keys[2], 64)

var img image.Image

var imgBytes []byte

if cache[key] != nil {

fmt.Println("/1")

imgBytes = cache[key]

fmt.Println("/2")

} else {

file, err := os.Open("2\_5467644889959236843.json")

if err != nil {

fmt.Println(err)

os.Exit(1)

}

defer file.Close()

fmt.Println("/")

data := make([]byte, 64)

raw := ""

for {

r, err := file.Read(data)

if err == io.EOF { // если конец файла

break // выходим из цикла

}

raw = raw + string(data[:r])

}

rawGeometryJSON := []byte(raw)

fmt.Println("/")

if img, err = getPNG(rawGeometryJSON, z, x, y); err != nil {

fmt.Println(err.Error())

}

fmt.Println("/")

buffer := new(bytes.Buffer) //buffer - \*bytes.Buffer

png.Encode(buffer, img) //img - image.Image

imgBytes = buffer.Bytes()

cache[key] = imgBytes

}

fmt.Println("/")

w.Write(imgBytes)

}

func main() {

cache = make(map[string][]byte, 0)

http.Handle("/assets/", http.StripPrefix("/assets/", http.FileServer(http.Dir("./assets/"))))

http.HandleFunc("/", indexHandler)

http.HandleFunc("/tile/", draw)

http.ListenAndServe(":3000", nil)

}

func getPNG(featureCollectionJSON []byte, z float64, x float64, y float64) (image.Image, error) {

var coordinates [][][][][]float64

var err error

if coordinates, err = getMultyCoordinates(featureCollectionJSON); err != nil {

return nil, err

}

dc := gg.NewContext(width, height)

scale := 1.0

dc.InvertY()

//рисуем полигоны

forEachPolygon(dc, coordinates, func(polygonCoordinates [][]float64) {

dc.SetRGB(rand.Float64(), rand.Float64(), rand.Float64())

drawByPolygonCoordinates(dc, polygonCoordinates, scale, dc.Fill, z, x, y)

})

//рисуем контуры полигонов

dc.SetLineWidth(2)

forEachPolygon(dc, coordinates, func(polygonCoordinates [][]float64) {

dc.SetRGB(rand.Float64(), rand.Float64(), rand.Float64())

drawByPolygonCoordinates(dc, polygonCoordinates, scale, dc.Stroke, z, x, y)

})

out := dc.Image()

return out, nil

}

func getMultyCoordinates(featureCollectionJSON []byte) ([][][][][]float64, error) {

var featureCollection \*geojson.FeatureCollection

var err error

if featureCollection, err = geojson.UnmarshalFeatureCollection(featureCollectionJSON); err != nil {

return nil, err

}

var features = featureCollection.Features

var coordinates [][][][][]float64

for i := 0; i < len(features); i++ {

coordinates = append(coordinates, features[i].Geometry.MultiPolygon)

}

return coordinates, nil

}

func forEachPolygon(dc \*gg.Context, coordinates [][][][][]float64, callback func([][]float64)) {

for i := 0; i < len(coordinates); i++ {

for j := 0; j < len(coordinates[i]); j++ {

callback(coordinates[i][j][0])

}

}

}

const mercatorMaxValue float64 = 20037508.342789244

const mercatorToCanvasScaleFactorX = float64(width) / (mercatorMaxValue)

const mercatorToCanvasScaleFactorY = float64(height) / (mercatorMaxValue)

func drawByPolygonCoordinates(dc \*gg.Context, coordinates [][]float64, scale float64, method func(), z float64, xTile float64, yTile float64) {

scale = scale \* math.Pow(2, z)

dx := float64(dc.Width())\*(xTile) - 138.5\*scale

dy := float64(dc.Height())\*(math.Pow(2, z)-1-yTile) - 128\*scale

for index := 0; index < len(coordinates)-1; index++ {

x, y := mercator.LatLonToMeters(coordinates[index][1], convertNegativeX(coordinates[index][0]))

x, y = centerRussia(x, y)

x \*= mercatorToCanvasScaleFactorX \* scale \* 0.5

y \*= mercatorToCanvasScaleFactorY \* scale \* 0.5

x -= dx

y -= dy

dc.LineTo(x, y)

}

dc.ClosePath()

method()

}

func centerRussia(x float64, y float64) (float64, float64) {

var west = float64(1635093.15883866)

if x > 0 {

x -= west

} else {

x += 2\*mercatorMaxValue - west

}

return x, y

}

func convertNegativeX(x float64) float64 {

if x < 0 {

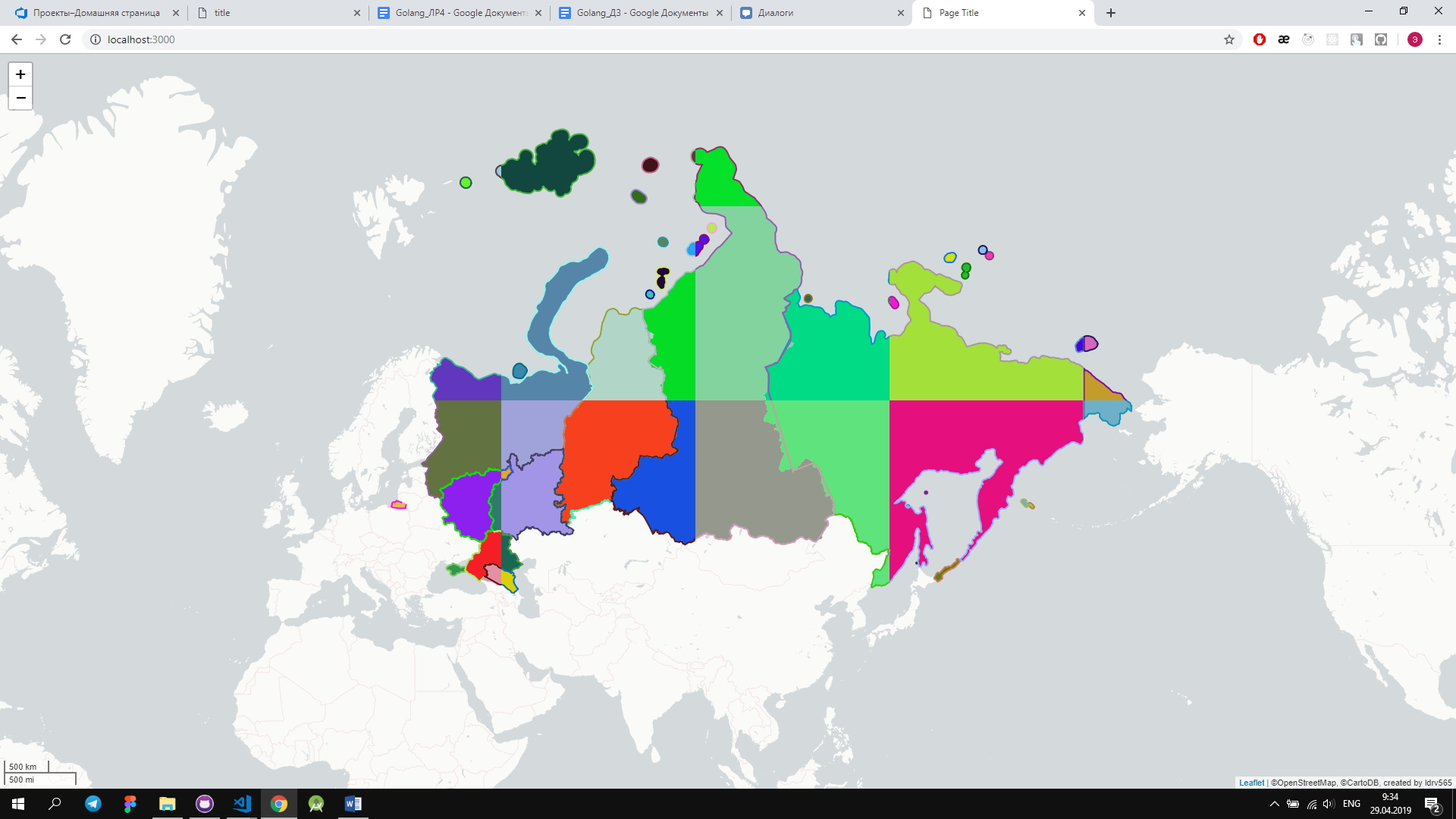
x = x - 360

}

return x

}

### Результат выполнения программы:



### Вывод:

В результате выполнения данной лабораторной работы было произведено знакомство с понятием семантики объектов, знакомство с проблемами изображения метаданных.