Сервер приложений

Содержание

[Введение 3](#_Toc52268751)

[1 Технические характеристики 4](#_Toc52268752)

[1.1 Среда разработки 4](#_Toc52268753)

[2 Описание программы 5](#_Toc52268754)

[2.1 Сторонние библиотеки 5](#_Toc52268755)

[2.2 Структура RoleAccess.json 6](#_Toc52268756)

[2.3 Настроечный файл Config.toml 6](#_Toc52268757)

[3 Ресурсы 8](#_Toc52268758)

[3.1 Websocket 8](#_Toc52268759)

[3.2.1 Основное окно карты 8](#_Toc52268760)

[3.2.2 Режим Зеленая Улица 10](#_Toc52268761)

[3.2.3 Управление перекрестком 11](#_Toc52268762)

[3.2.4 Управление привязкой 12](#_Toc52268763)

# Введение

Сервер приложений (TLServer) осуществляет обработку запросов от пользователей.

Данные передаются посредством протокола HTTP (HyperText Transfer Protocol - протокол передачи гипертекста) и WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

Хранение информации осуществляется посредством  свободной объектно-реляционная системы управления базами данных PostgreSQL, базируется на языке SQL и поддерживает многие из возможностей стандарта SQL.

Сервер приложений осуществляет связь посредством запросов с сервером устройств Ag-Server.

1 Технические характеристики

* 1. Среда разработки

Языком программирования выбран GOlang - компилируемый многопоточный язык программирования, разработанный внутри компании Google.

Версия языка на 09.2020 является 1.15.

# 2 Описание программы

## 2.1 Сторонние библиотеки

1) github.com/BurntSushi/toml

Библиотека для работы с TOML (формат конфигурационных файлов), позволяется кодировать и раскодировать файлы данного формата.

2) github.com/dgrijalva/jwt-go

Библиотека поддерживает синтаксический анализ и проверку, а также генерацию и подписание JWT (json web token). В настоящее время поддерживаются следующие алгоритмы подписи: HMAC SHA, RSA, RSA-PSS и ECDSA.

3) github.com/gorilla/websocket

Библиотека обработки протокола связи websocket – содержит в себе набор инструментов, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.

4) github.com/gin-gonic/gin

Библиотека реализует маршрутизатор и диспетчер запросов для сопоставления входящих запросов с их соответствующим обработчиком.

5) github.com/jmoiron/sqlx

Библиотека реализует интерфейсы для работы с БД.

6) github.com/lib/pq

Драйвер для работы с postgresql.

7) golang.org/x/crypto

Стандартная библиотека, предоставляющая возможности по шифрованию информации.

8) github.com/ruraomsk/ag-server

Сервер управления АСУД.

9) github.com/jordan-wright/email

Библиотека, предоставляющая инструменты для работы с почтовыми клиентами.

10) github.com/go-ozzo/ozzo-validation

Библиотека, предоставляющая инструментарий для проверки данных.

## 2.2 Структура RoleAccess.json

RoleAccess.json содержит информацию для осуществления контроля доступа пользователей к ресурсам.

Файл содержит 3 ключа:

1. roles – информация о роли

ключи:

name – имя,

permissions - массив разрешений;

1. permissions – информация о разрешении

ключи:

id – уникальный номер,

description – описание,

visible - флаг видимости, необходим для работы с гибкими ролями;

1. routes – информация о маршруте

ключи:

description – описание,

permission – к какому разрешению относится маршрут,

path – маршрут ресурса.

## 

## 2.3 Настроечный файл Config.toml

Config.toml содержит всю настроечную информацию для корректной работы сервера.

Конфигурационный файл (config.toml) для TLServer.

logger\_path = "logfiles"  
static\_path = "resources/static"  
web\_path = "resources/web"  
free\_path = "resources/free"  
ssl\_path = "ssl"  
server\_ip = ":8082"  
  
#настройка подключения к тсп серверу обемена командами  
[tcpServer]  
tcpServerAddress = "192.168.115.115"  
portState = "2002"  
portArmCommand = "2001"  
portChangeProtocol = "2003"  
portGPS = "1093"  
  
#настройки для постгреса  
[database]  
db\_name = "agserv"  
db\_password = "162747"  
db\_user = "postgres"  
db\_type = "postgres"  
db\_host = "192.168.115.115"  
db\_port = "5432"  
db\_SetMaxOpenConst = 50  
db\_SetMaxIdleConst = 10

Содержит поля:

1. Общая группа:
2. logger\_path –указывается путь до папки хранения файлов лога сервера;
3. static\_path - указывается путь до папки static в которой содержатся ресурсы для заполнения страниц;
4. web\_path - указывается путь до папки web с ресурсами для загрузки страниц (css, html, js);
5. free\_path - указывается путь до папки free с общими ресурсами для загрузки страниц (css, html, js);
6. server\_ip - указывается адрес и порт на котором будет запушен сервер;
7. Группа настройки подключения к ТСП серверу (“[tcpServer]”):
8. tcpServerAddress - указывается адрес сервера ag-server;
9. portState - указывается порт для обмена State пакетами;
10. portArmCommand - указывается порт для обмена сообщениями управления устройствами;
11. portChangeProtocol - указывается порт для обмена сообщениями смены протокола;
12. portGPS - указывается порт GPS соединения для отображения пользователю;
13. Группа настройки подключения к Базе Данных (“[database]”):
14. db\_name – указывается название базы данных;
15. db\_password – указывается пароль для подключения к базе данных;
16. db\_user – указывается login пользователя для подключения к базе данных;
17. db\_type – указывается тип базы данных;
18. db\_host – указывается адрес для подключения к базе данных;
19. db\_port – указывается порт для подключения к базе данных (стандартный порт 5432);
20. db\_SetMaxOpenConst – указывается максимальное количество одновременных подключений к базе данных;

8. db\_SetMaxIdleConst – указывается максимальное количество одновременных подключений к базе данных в режиме ожидания;

3 Ресурсы

Все URL в начале имеют ip и порт (пример 192.168.1.1:8080/), также зарезервированное слово в URL – :slug – после авторизации пользователя заменяется на имя его учетной записи.

3.1 Websocket

3.2.1 Основное окно карты

URL: /map или /

Открытие сокета для данного ресурса /mapW

Сокет реализует контроль и отправку каждому пользователю списка перекрестков и их состояний, аутентификацию пользователя в системе.

Обмен сообщениями происходит структурой mapResponse:

type mapResponse struct {

Type string `json:"type"` //тип сообщения

Data map[string]interface{} `json:"data"` //данные

}

Поле Type в запросе:

1. login

Принимает данные с полями login и password

type Data struct {

Login string `json:"login"` //Имя пользователя

Password string `json:"password"` //Пароль

}

Возвращает ответ об авторизации пользователя, в случае неудачи возвращается информация об ошибке. При успешной авторизация пользователю формируется информация для дальнейшей работы в системе, формируется токен для доступа к другим ресурсам (токен записывается в бд для контроля уникальности):

type Data struct {

Token string `json:"token"` //токен доступа пользователя

Access []string `json:"access"` //доступ к ресурсам

Area string `json:"area"` //район пользователя

Region string `json:"region"` //регион пользователя

Role string `json:"role"` //роль пользователя

Box BoxPoint `json:"boxPoint"` //зона отображения карты

AuthorizedFlag bool `json:"authorizedFlag"` //флаг авторизации

Description string `json:"description"` //описание арм пользователя

AreaInfo areaInfo `json:"areaInfo"` //информация о районе с включенными в него подрайонами

AreaZone areaZone `json:"areaZone"` //информация о районах c

координатами для перемещения

RegionInfo regInfo `json:"regionInfo"` //информация о регионе

TF []TFLight `json:"tflight"` //список всех светофоров

}

1. logOut

В поле Data данные не нужны. По данному запросу происходит выход пользователя из системы и удаления токена из БД.

1. changeAcc

Принимает данные с полями login и password

type Data struct {

Login string `json:"login"` //Имя пользователя

Password string `json:"password"` //Пароль

}

При смене пользователя происходит сначала авторизация пользователя с формированием данных для работы нового пользователя см. пункт 1, при успешной авторизации нового пользователя старый пользователь проходит процедуру выхода.

1. jump

Перемещение по карте, принимает структуру location

type Locations struct {

Region string `json:"region"` //регион

Area []string `json:"area"` //районы

}

По данному запросу формируют координаты выбранной области

type BoxPoint struct {

Point0 Point `json:"point0"` //левая нижняя точка на карте

Point1 Point `json:"point1"` //правая верхняя точка на карте

}

1. checkConn

В поле Data данные не нужны. По данному запросу происходит проверка соединения с БД и с Сервером управления.

В автоматическом режиме каждые 5 секунд сокет контролирует изменение перекрестков и отправляется изменение каждому пользователю. Отправляется список измененных перекрестков и тип tflight. Структура имеет вид:

type mapResponse struct {

Type string `json:"type"` //тип сообщения

Data map[string][]data.TrafficLights `json:"data"` //данные

}

TrafficLights информация о светофоре

type TrafficLights struct {

ID int `json:"ID"` //Уникальный ID светофора

Region RegionInfo `json:"region"` //Регион

Area AreaInfo `json:"area"` //Район

Subarea int `json:"subarea"` //ПодРайон

Idevice int `json:"idevice"` //Реальный номер устройства

Sost TLSostInfo `json:"tlsost"` //Состояние светофора

Description string `json:"description"` //Описание светофора

Phases []int `json:"phases"` //Доступные фазы

Points locations.Point `json:"points"` //Координата где находится светофор

}

### 3.2.2 Режим Зеленая Улица

URL: /user/:slug/greenStreet

Открытие сокета для данного ресурса /user/:slug/greenStreetW

Сокет реализует контроль и отправку перекрестков каждому авторизированному пользователю, возможность создания, редактирования, удаления и управления маршрутами.

Обмен сообщениями происходит структурой gSResponse:

type gSResponse struct {

Type string `json:"type"` //тип сообщения

Data map[string]interface{} `json:"data"` //данные

}

Поле Type в запросе:

1. createRoute

Создает маршрут для режима ЗУ, принимает структуру Route

type Route struct {

Id int `json:"id"` //уникальный номер в бд

Region string `json:"region"` //регион

Description string `json:"description"` //описание маршрута

Box locations.BoxPoint `json:"box"` //координаты на которые перемещаться при выборе маршрута

List []RouteTL `json:"listTL"` //список светофоров входящих в маршрут

}

Если при создании маршрута возникла ошибка пользователю возвращается ошибка.

1. updateRoute

Изменение/редактирование маршрута, принимает структуру Route. Если при создании маршрута возникла ошибка пользователю возвращается ошибка.

1. deleteRoute

Удаление маршрута, принимает структуру Route. Если при создании маршрута возникла ошибка пользователю возвращается ошибка.

1. jump

Перемещение по карте, принимает структуру location

type Locations struct {

Region string `json:"region"` //регион

Area []string `json:"area"` //районы

}

По данному запросу формируют координаты выбранной области

type BoxPoint struct {

Point0 Point `json:"point0"` //левая нижняя точка на карте

Point1 Point `json:"point1"` //правая верхняя точка на карте

}

1. dispatch

Отправка на сервер управления команды для управления перекрестком принимает структуру CommandARM

type CommandARM struct {

ID int `json:"id"` //уникальный id устройства

User string `json:"user"` //пользователь который отправил команду

Command int `json:"cmd"` //команда

Params int `json:"param"` //параметр

}

В автоматическом режиме каждые 5 секунд сокет контролирует изменение перекрестков и отправляется изменение каждому пользователю. Отправляется список измененных перекрестков и тип tflight.

Каждую минуту проверяет состояние токена пользователя для контроля сеанса, если время действия токена истекло, сокет автоматически закрывается.

### 3.2.3 Управление перекрестком

URL: /user/:slug/cross

Открытие сокета для данного ресурса /user/:slug/crossW

Сокет реализует контроль и управление перекрестком для каждого авторизированного пользователя.

Обмен сообщениями происходит структурой crossResponse:

type crossResponse struct {

Type [string](http://localhost:6060/pkg/builtin/#string) `json:"type"`

Data map[[string](http://localhost:6060/pkg/builtin/#string)]interface{} `json:"data"`

}

Поле Type в запросе:

1) dispatch

Отправка на сервер управления команды для управления перекрестком принимает структуру CommandARM

type CommandARM struct {

ID int `json:"id"` //уникальный id устройства

User string `json:"user"` //пользователь который отправил команду

Command int `json:"cmd"` //команда

Params int `json:"param"` //параметр

}

В автоматическом режиме каждую секунду сокет контролирует описание перекрестка отправляя тип (crossUpdate) и фазу (phase).

Каждую минуту проверяет состояние токена пользователя для контроля сеанса, если время действия токена истекло, сокет автоматически закрывается.

3.2.4 Управление привязкой

URL: /user/:slug/cross/control

Открытие сокета для данного ресурса /user/:slug/cross/controlW

Сокет реализует контроль данных привязки устройства, создание привязки, выбор координат расположения устройства, отправку привязки на устройства.

Обмен сообщениями происходит структурой ControlSokResponse:

type ControlSokResponse struct {

Type [string](http://localhost:6060/pkg/builtin/#string) `json:"type"`

Data map[[string](http://localhost:6060/pkg/builtin/#string)]interface{} `json:"data"`

}

Полу Type в запросе:

1. checkB

Проверка привязки (pudge.Cross) на корректность заполнения. Пользователю возвращается результат проверки с указанием места ошибки и рекомендованного варианта решения.

1. sendB

Отправка измененной привязки (pudge.Cross) на сервер (Ag-Server) на исполнение, если пользователь указывает новые координаты расположения перекрестка создается новая подложка (снимок карты) перекреста.

1. createB

Создание новой привязки (перекрестка) на сервер (Ag-Server) отправляется новая привязка (pudge.Cross), по указанным координатам формируется подложка (снимок карты) перекрестка.

1. deleteB

Удаление привязки (перекрестка) на сервер (Ag-Server) отправляется команда удаления привязки.

1. updateB

Обновление данных привязки из базы данных.

1. editInfoB

Запрос списка пользователей, которые в данный момент редактируют/просматривают данную привязку.

1. dispatch

Отправка на сервер управления команды для управления перекрестком принимает структуру CommandARM.

type CommandARM struct {

ID int `json:"id"` //уникальный id устройства

User string `json:"user"` //пользователь который отправил команду

Command int `json:"cmd"` //команда

Params int `json:"param"` //параметр

}

В автоматическом режиме сокет контролирует пользователей, которые редактирует и просматривают данное устройство, если пользователь редактирующий перекресток закончил работу, то признак редактирования отдается одному из пользователей которые просматривали устройство.

Каждую минуту проверяет состояние токена пользователя для контроля сеанса, если время действия токена истекло, сокет автоматически закрывается.

3.2.5 Технологические режимы

URL: /user/:slug/techArm

Открытие сокета для данного ресурса /user/:slug/techArmW

Сокет реализует сбор информации о всех перекрестках и подключенных устройствах, отправка актуальной информации об изменении состояния устройства, управление устройствами.

Обмен сообщениями происходит структурой armResponse:

type armResponse struct {

Type [string](http://localhost:6060/pkg/builtin/#string) `json:"type"` //тип сообщения

Data map[[string](http://localhost:6060/pkg/builtin/#string)]interface{} `json:"data"` //данные

}

Поле Type в запросе:

1. dispatch

Отправка на сервер управления команды для управления перекрестком принимает структуру CommandARM.

type CommandARM struct {

ID int `json:"id"` //уникальный id устройства

User string `json:"user"` //пользователь который отправил команду

Command int `json:"cmd"` //команда

Params int `json:"param"` //параметр

}

1. gprs

Отправка на сервер управления команды для изменения настроек GPRS протокола (структура сообщения comm.ChangeProtocol).

Каждую минуту проверяет состояние токена пользователя для контроля сеанса, если время действия токена истекло, сокет автоматически закрывается.

3.2.6 Управление по характерным точкам

URL: /user/:slug/charPoints

Открытие сокета для данного ресурса /user/:slug/charPointsW

Сокет реализует отправку пользователю сортированного массива перекрестков, создание, удаление и изменение планов управления.

Обмен сообщениями происходит структурой armResponse:

type armResponse struct {

Type [string](http://localhost:6060/pkg/builtin/#string) `json:"type"` //тип сообщения

Data map[[string](http://localhost:6060/pkg/builtin/#string)]interface{} `json:"data"` //данные

}

Поле Type в запросе:

1. xctrlCreate