

Программная реализация умной теплицы на базе ESP-32

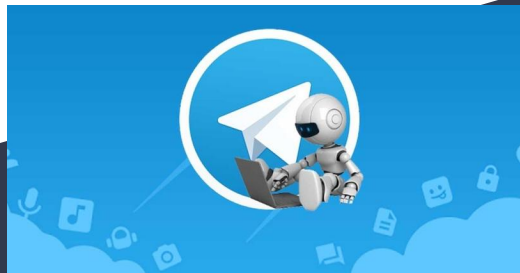
команда Орешник

Зависимости проекта pom.xml и иерархия файлов


```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.7.3</version>
  </dependency>
  <dependency>
    <groupId>org.telegram</groupId>
    <artifactId>telegrambots</artifactId>
    <version>6.8.0</version>
  </dependency>
  <dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20210307</version>
  </dependency>
</dependencies>
```

```
▼ components
  ● CheckRoot
  ● DateTime
  ● MyComponent
▼ controllers
  ● ControllerWeb
▼ dop
  ▼ smart
    ● Smart
    ● Ssvet
    ● Stemp
    ● Svlashnost
    ● Esp32Client
    ● GrafickJson
    ● SmartStatic
    ● SmartUpravESP
▼ models
  ● Post
▼ repo
  ● PostRepository
▼ telegram
  ● NastroiBot
  ● StartBot
  ● Terminator
  ● MeteoStarSpringApplication
▼ resources
  ▼ static
    ▼ templates
      ● autorisation.html
      ● full.html
      ● grafik.html
      ● index.html
      ● smart.html
      ● statica.html
    ● application.properties
```

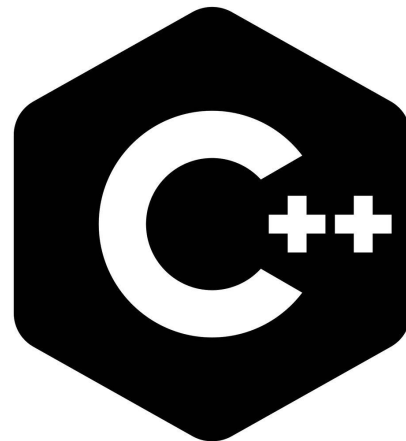
Основные библиотеки и фреймворки



Основные языки

The logo for JavaScript, featuring the letters 'JS' in a bold, black, sans-serif font centered on a solid yellow square background.

JS



Подключение к БД, создание CRUD системы

```
@Entity
public class Post {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private double vlashnost, temp;
    private String data;
    private int svet;

    public Long getId(){return id;}

    public void setId(Long id){this.id = id;}

    public int getSvet(){return svet;}

    public void setSvet(int svet){this.svet = svet;}

    public double getVlashnost(){return vlashnost;}

    public void setVlashnost(double vlashnost){this.vlashnost = vlashnost;}

    public double getTemp(){return temp;}

    public void setTemp(double temp){this.temp = temp;}

    public String getData(){return data;}

    public void setData(String data){this.data = data;}

    public Post(int svet, double vlashnost, double temp, String data) {
        this.svet = svet;
        this.vlashnost = vlashnost;
        this.temp = temp;
        this.data = data;
    }

    public Post() {
    }

    this.vlashnost =
    this.temp =
    this.data =

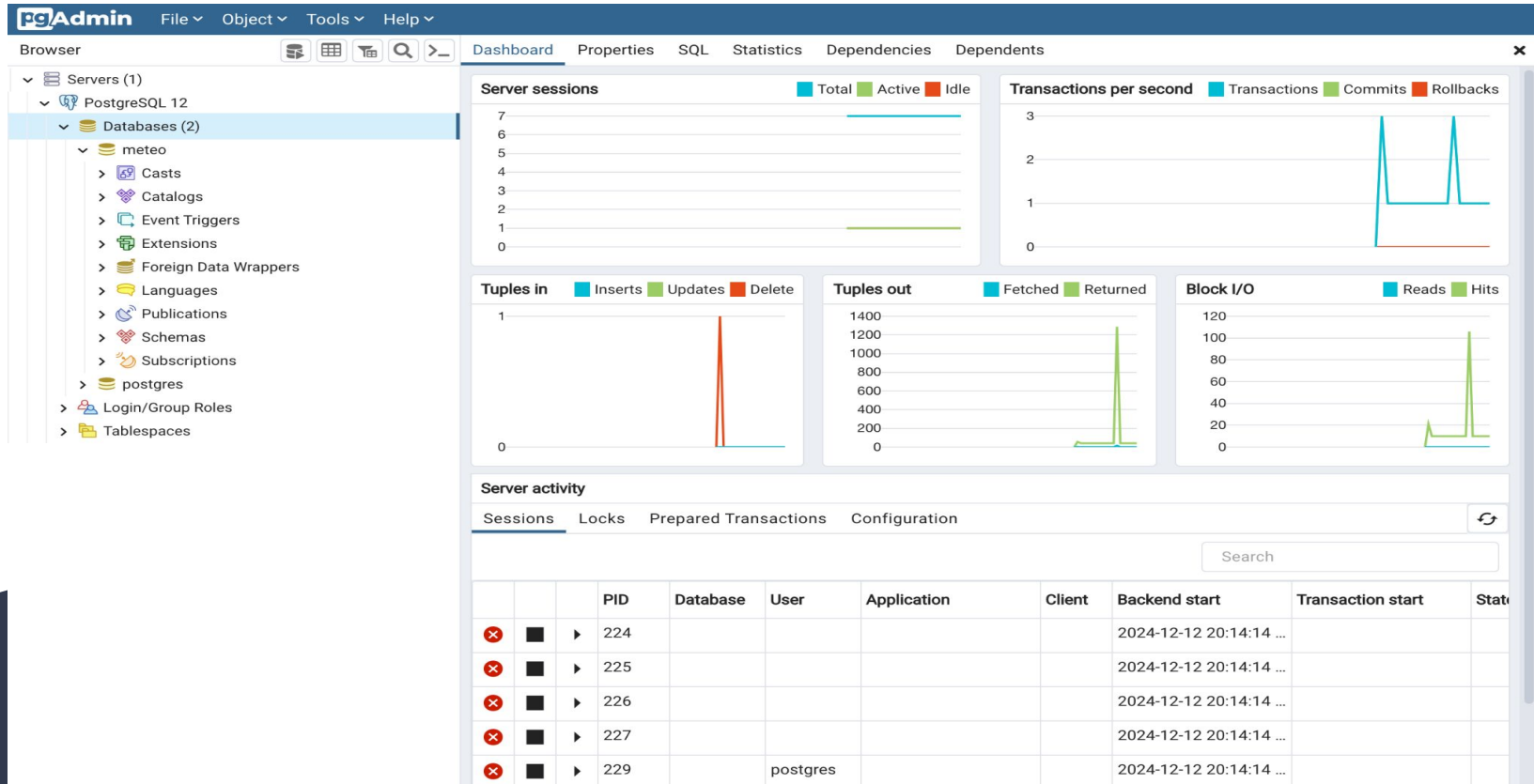
    public Post() { public interface PostRepository extends CrudRepository<Post, Long> { }
```

```
ort com.example.MeteoStarSpring.models.Post;
```

```
this.vlashnost =
this.temp =
this.data = import com.example.MeteoStarSpring.models.Post;
import org.springframework.data.repository.CrudRepository;
```

```
public Post() { public interface PostRepository extends CrudRepository<Post, Long> { }
```

Инициализация БД, используя СУБД PostgreSQL



Подключение к БД, настройка root доступа, настройка esp32

```
public class NastroiBot {  
  
    private final static String botUsername = "realteplotbot";  
    private final static String botToken = "7706868010:AAF5rpDc7LLLtue8bK6o9KPE5BRWCSe4eq4";  
    private final static String allowedUserId = "793730544";  
    private final static String ip = "192.168.2.45";  
    private final static String adminToken = "/admin/token/reiojifjoiwr38923e0u58turwkjowHFI733jwf023/";  
  
    public static String getBotUsername() { return botUsername; }  
  
    public static String getBotToken() { return botToken; }  
  
    public static String getAllowedUserId() { return allowedUserId; }  
  
    public static String getIp() {  
        return ip;  
    }  
  
    public static String getAdminToken() { return adminToken; }  
}
```

```
spring.application.name=MeteorStarSpring
```

```
spring.datasource.url=jdbc:postgresql://localhost:5432/meteor
```

```
spring.datasource.username=postgres
```

```
spring.datasource.password=postgres
```

```
spring.datasource.driver-class-name=org.postgresql.Driver
```

```
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

```
my.ipesp=192.168.2.45
```

```
my.token=/admin/token/reiojifjoiwr38923e0u58turwkjowHFI733jwf023/
```

```
my.log=admin
```

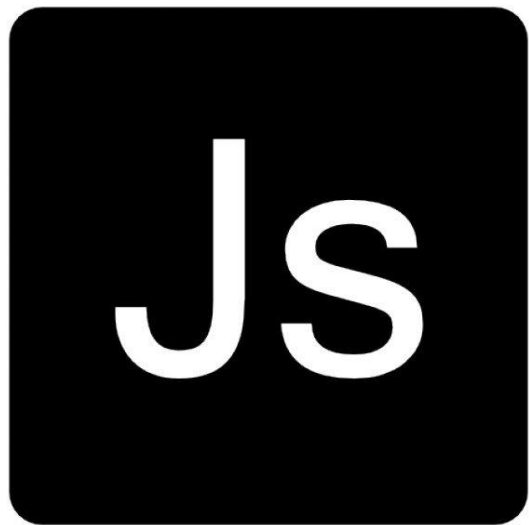
```
my.pass=admin
```

Создание RESTful сервиса и запуск телеграм бота в отдельном потоке

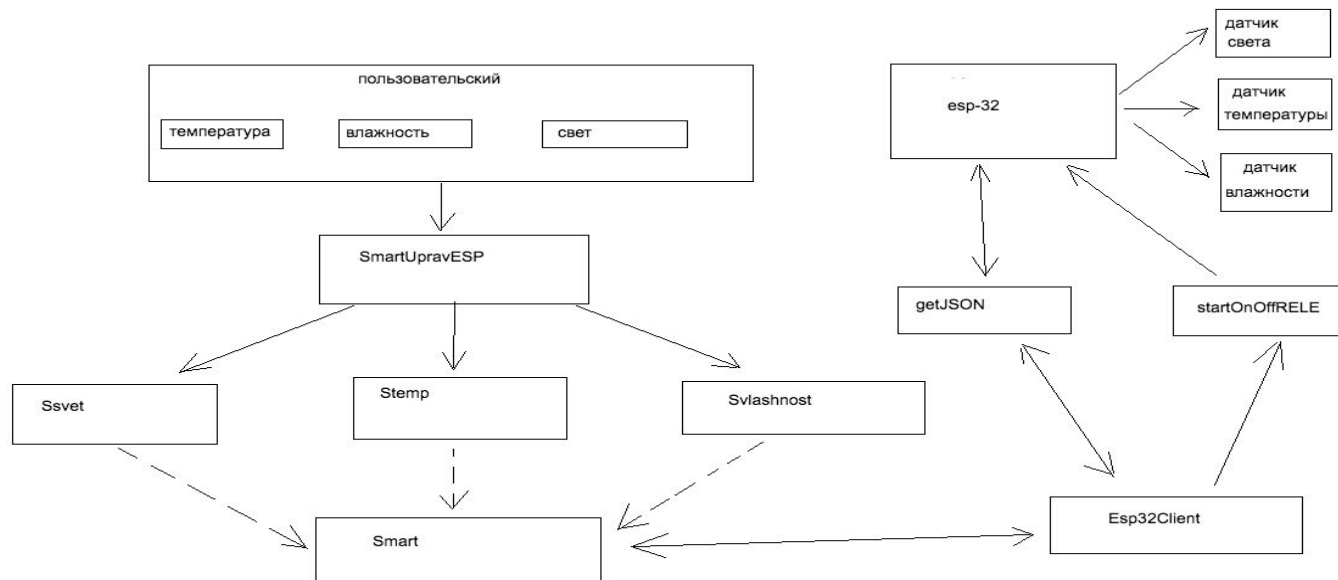
```
public class StartBot implements Runnable {  
  
    @Override  
    public void run() {  
        try {  
            TelegramBotsApi botsApi = new TelegramBotsApi(DefaultBotSession.class);  
            botsApi.registerBot(new Terminator());  
        } catch (TelegramApiException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
@Controller  
public class ControllerWeb {  
  
    @Autowired  
    private PostRepository postRepository;  
    @Autowired  
    CheckRoot croot;  
    @Autowired  
    MyComponent myComp;  
    @Autowired  
    GrafickJson grafick;  
  
    private boolean root = false;  
  
    @GetMapping("/")  
    public String autorisation(Model model) {  
        model.addAttribute("title", "Вход");  
        if (!root) {  
            return "autorisation";  
        } else {  
            return "redirect:/glav";  
        }  
    }  
  
    @PostMapping("/autoisation")  
    public String trueAutorisation(@RequestParam String log, @RequestParam String pass, Model model) {  
        if (croot.checkRoot(log, pass)) {  
            root = true;  
            return "redirect:/glav"; // либо index  
        } else {  
            return "autorisation";  
        }  
    }  
}
```


Отображение на стороне клиента



Паттерн Наблюдатель, многопоточная реализация умного ухода



Код:

```
public class Smart {

    void proverka(double norms_temp_double, String esp_temp, String smart_time, String ip, String token,
                  String num_rele) {
        Esp32Client esp = new Esp32Client();
        // esp.Esp32GetJson(ip, token);
        if (Double.parseDouble(esp_temp) < norms_temp_double) {
            int dop = Integer.parseInt(smart_time) * 1000;

            // отправляем http запрос
            System.out.println("Включаем реле - "+num_rele+" на "+dop+" секунд!");
            esp.startReleOnOff(ip, token, num_rele, work: "2");
            try {
                Thread.sleep(dop);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            esp.startReleOnOff(ip, token, num_rele, work: "1");
            System.out.println("Выключаем реле - "+num_rele);
        }
    }
}
```

```
public class Ssvet extends Smart implements Runnable {

    private double norms_double;
    private String esp_temp, smart_time, ip, token, num_rele;

    public Ssvet(double norms_double, String esp_temp, String smart_time, String ip, String token, String num_rele) {
        this.norms_double = norms_double;
        this.esp_temp = esp_temp;
        this.smart_time = smart_time;
        this.ip = ip;
        this.token = token;
        this.num_rele = num_rele;
    }

    @Override
    public void run() { proverka(norms_double, esp_temp, smart_time, ip, token, num_rele); }
}
```

```
public class Svlashnost extends Smart implements Runnable {

    private double norms_double;
    private String esp_temp, smart_time, ip, token, num_rele;

    public Svlashnost(double norms_double, String esp_temp, String smart_time, String ip, String token, String num_rele) {
        this.norms_double = norms_double;
        this.esp_temp = esp_temp;
        this.smart_time = smart_time;
        this.ip = ip;
        this.token = token;
        this.num_rele = num_rele;
    }

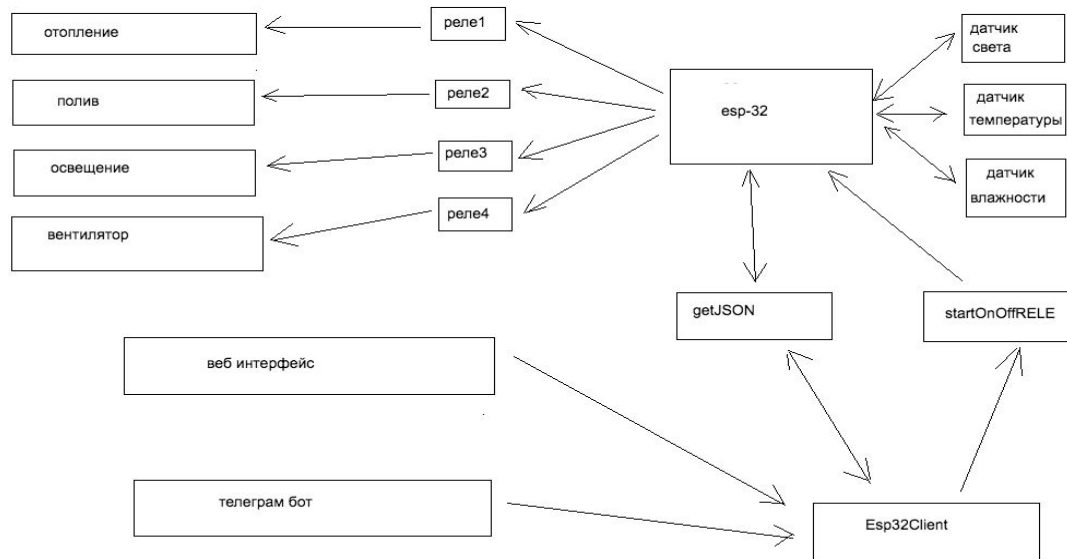
    @Override
    public void run() { proverka(norms_double, esp_temp, smart_time, ip, token, num_rele); }
}
```

Многопоточный сбор статистики датчиков с построением Графов

```
public class SmartStatic implements Runnable {  
  
    private int time;  
    private String ip, token;  
  
    public SmartStatic(int time, String ip, String token) {  
        this.time = time;  
        this.ip = ip;  
        this.token = token;  
    }  
  
    @Override  
    public void run() { sbor(); }  
  
    private void sbor() {  
        for (;;) {  
            Esp32Client client = new Esp32Client();  
            client.Esp32GetJson(ip, token);  
            try {  
                Thread.sleep(millis: time * 1000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

```
@Autowired  
private PostRepository postRepository;  
  
public String generate(String name_dannye) {  
    ArrayList<Double> xData = new ArrayList<>();  
    ArrayList<Double> yData = new ArrayList<>();  
  
    // Получаем все ID  
    Iterable<Post> allPosts = postRepository.findAll();  
    for (Post post : allPosts) {  
        xData.add(Double.valueOf(post.getId()));  
    }  
  
    // Получаем значения из базы по указанной строке  
    Iterable<Post> filteredPosts = postRepository.findAll(); // Получаем все записи  
    for (Post post : filteredPosts) {  
        if (name_dannye.equals("temp")) {  
            yData.add(post.getTemp());  
        } else if (name_dannye.equals("vlashnost")) {  
            yData.add(post.getVlashnost());  
        } else if (name_dannye.equals("svet")) {  
            yData.add(Double.valueOf(post.getSvet()));  
        }  
    }  
  
    return generateJSON(xData, yData);  
}  
  
private String generateJSON(ArrayList<Double> xData, ArrayList<Double> yData) {  
    StringBuilder sb = new StringBuilder();  
    sb.append("{ \"x\": [");  
    for (Double x : xData) {  
        sb.append(x).append(", ");  
    }  
    sb.delete(sb.length() - 2, sb.length()).append("], \"y\": [");  
    for (Double y : yData) {  
        sb.append(y).append(", ");  
    }  
    sb.delete(sb.length() - 2, sb.length()).append("]]");  
    return sb.toString();  
}
```

Java приложение и ESP32. Общение используя HTTP и локальный сервер



Код:

```
public void startReleOnOff(String ip, String token, String num_rele, String work) { // 1-off 2-on
    String urlString = "http://" + ip + token + "rele" + num_rele + "?work=" + work;

    try {
        URL url = new URL(urlString);

        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");

        int responseCode = connection.getResponseCode();
        System.out.println("Response Code: " + responseCode);

        if (responseCode == HttpURLConnection.HTTP_OK) {
            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()))
            String inputLine;
            StringBuilder response = new StringBuilder();

            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();

            if (Integer.parseInt(work) == 1) {
                System.out.println("Rele " + num_rele + " turned OFF");
            } else {
                System.out.println("Rele " + num_rele + " turned ON");
            }
        }
    } catch (Exception e) {
        System.out.println("Error smart rele - " + e);
    }
}
```

```
public void Esp32GetJson(String ip, String token) {
    // Пример: http://192.168.1.100/admin/token/reiojfjoiwr38923e0u58turwkjowHF1733jwf023/
    String urlString = "http://" + ip + token;

    try {
        URL url = new URL(urlString);
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod("GET");

        int responseCode = connection.getResponseCode();
        System.out.println("Response Code: " + responseCode);

        if (responseCode == HttpURLConnection.HTTP_OK) {
            BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
            String inputLine;
            StringBuilder response = new StringBuilder();

            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();

            String jsonResponse = response.toString();
            System.out.println("JSON Response: " + jsonResponse);

            JSONObject jsonObject = new JSONObject(jsonResponse);
            this.svet = jsonObject.optString(key: "light", defaultValue: "1");
            this.temp = jsonObject.optString(key: "temperature", defaultValue: "183");
            this.vlashnost = jsonObject.optString(key: "humidity", defaultValue: "6");

            System.out.println("GET request worked");
        } else {
            System.out.println("GET request NOT worked");
        }
    } catch (Exception e) {
        System.out.println("Error get json - " + e);
    }
}
```