

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1170

Usluga u FER-ovoj IoT platformi za alarmiranje u slučaju kvara uređaja

Valentina Valić

Zagreb, lipanj 2023.

Zahvaljujem se mentoru prof. dr. sc. Mariju Kušeku na pruženoj pomoći i savjetima što mi je uvelike pomoglo pri izradi završnog rada. Također se zahvaljujem autorima literature koju sam koristila jer su svoje radove i dostignuća učinili javno dostupnima za korištenje.

SADRŽAJ

1. Uvod	1
2. Pregled IoT platformi i alarmnih sustava	2
2.1. Koncept internet of things (IoT)	2
2.2. Pregled postojećih IOT platformi	5
2.3. Pregled alarmnih sustava u IoT okruženju	8
3. Spring Boot aplikacija za IoT alarmiranje	10
3.1. Opis Spring Boot frameworka	10
3.1.1. Uvod u Spring Boot	10
3.1.2. Karakteristike Spring Boot frameworka	10
3.1.3. Glavne komponente Spring Boot frameworka	13
3.2. Arhitektura aplikacije za IoT alarmiranje	15
3.3. Korištene tehnologije i alati	16
4. Generiranje alarma u Spring Boot aplikaciji	18
4.1. Potreba za generiranjem alarma	18
4.2. Spring Boot poslužitelj	18
4.3. Detekcija kvara uređaja u aplikaciji	19
4.4. Postavljanje i konfiguracija alarma u aplikaciji	20
4.5. Integracija s Microsoft Teamsom i Telegramom za slanje alarma	23
4.5.1. Kreiranje Telegram bota	23
4.5.2. Implementacija i integracija alarma putem Telegrama u aplikaciji	24
4.5.3. Kreiranje Microsoft Teams bota	26
4.5.4. Implementacija i integracija alarma putem Microsoft Teamsa u aplikaciji	28

5. Primjer korištenja aplikacije za IoT alarmiranje	30
5.1. Keycloak autentifikacija	30
5.2. Dodavanje, ažuriranje i brisanje alarma	31
5.3. Integracija s vanjskim servisima	37
6. Zaključak	39
Literatura	40

1. Uvod

U današnjem digitalnom dobu, Internet of Things (IoT) tehnologija igra ključnu ulogu u razvoju pametnih sustava i povezivanju uređaja diljem svijeta. Internet stvari (IoT) opisuje uređaje ugrađene s senzorima i aktuatorima koji komuniciraju s računalnim sustavima putem žičanih ili bežičnih mreža, omogućavajući digitalno praćenje ili čak kontrolu fizičkog svijeta [1]. Sve veći broj ovih uređaja koristi se za različite svrhe, od praćenja okoliša i upravljanja prometom do automatizacije kućanstava i industrijskih procesa.

U okviru Laboratorija za Internet stvari na Fakultetu elektrotehnike i računarstva (FER) razvijena je IoT platforma koja omogućuje povezivanje i upravljanje raznovrsnim IoT uređajima. Ova platforma omogućuje prikupljanje podataka s različitih senzora, njihovo pohranjivanje te pružanje mogućnosti dohvatanja podataka putem REST sučelja.

Međutim, ključno je osigurati da se bilo kakvi problemi, poput kvarova uređaja ili ispraznjene baterije, brzo otkriju i adekvatno adresiraju. U tu svrhu, cilj ovog istraživanja je nadograditi postojeću FER-ovu IoT platformu uslugom za alarmiranje u slučaju kvara uređaja.

Ova usluga će koristiti komunikacijske kanale kao što su Microsoft Teams i Telegram kako bi se osiguralo da relevantne osobe budu odmah obaviještene o mogućim problemima. Alarmi će biti generirani kada se detektira da određeni IoT uređaj nije poslao podatke tijekom određenog vremenskog razdoblja, što može ukazivati na kvar ili nestabilnost uređaja.

Ova nadogradnja FER-ove IoT platforme s uslugom za alarmiranje u slučaju kvara uređaja ima potencijal da značajno unaprijedi operativnu sigurnost i pouzdanost IoT sustava te pruži korisnicima mogućnost brzog i efikasnog reagiranja na eventualne probleme.

2. Pregled IoT platformi i alarmnih sustava

2.1. Koncept internet of things (IoT)

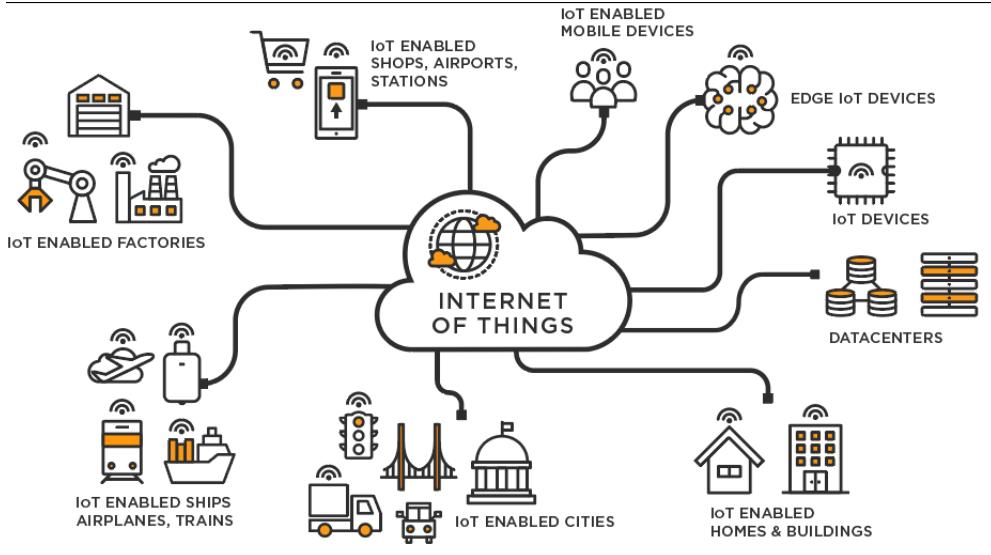
Posljednjih nekoliko godina, Internet stvari (IoT) postaje sve popularnija tema, iako koncept nije nov. Kevin Ashton postavio je temelje za IoT krajem prošlog stoljeća s ciljem unapređenja poslovanja kompanije Proctor & Gamble putem radiofrekventne identifikacije (RFID), bežičnog sustava sastavljenog od oznaka (eng. tag) i čitača [6]. Ashton je vjerovao da ako svi objekti koje koristimo u svakodnevnom životu budu imali identifikatore i mogućnost povezivanja na Internet, ti objekti će moći međusobno komunicirati i omogućiti nam upravljanje putem računala.

Ključna pitanja tada su bila kako povezati sve objekte putem Interneta, kako bi trebala izgledati bežična komunikacija i struktura Interneta koja bi mogla podržati povezivanje velikog broja objekata i njihovu komunikaciju. Osim toga, postavljalo se i pitanje napajanja i pogona većine tih stvari.

Danas, su ta pitanja riješena. Protokol IPv6 omogućava adresiranje i komuniciranje milijardi uređaja na mreži, a veličina i cijena bežične komunikacije su minimalne u odnosu na prošlost. Brzine Interneta su također dovoljno visoke da mogu podržati komunikaciju većeg broja uređaja. Uz to, baterije postaju sve izdržljivije [3].

Dakle, IoT predstavlja mrežnu infrastrukturu u kojoj fizičke i virtualne "stvari" svih vrsta komuniciraju i integrirane su nevidljivo. On opisuje sustav u kojem će objekti u fizičkom svijetu imati senzore, biti spojeni na Internet i dijeliti informacije s računalima i drugim uređajima na mreži. Koncept IoT-a prikazuje slika (Slika 2.1).

Ova napredna infrastruktura omogućuje nam, povezivanje industrijskih sustava, optimizaciju prometa i transporta, unaprjeđenje zdravstvene skrbi te brojne druge primjene. Sustavi IoT-a omogućuju nam prikupljanje i analizu velike količine podataka, što otvara nove mogućnosti za poboljšanje našeg svakodnevnog života [2, 5].



Slika 2.1: Koncept IoT-a [29]

Četiri su glavne komponente IOT-a [4]:

1. Senzori i aktuatori

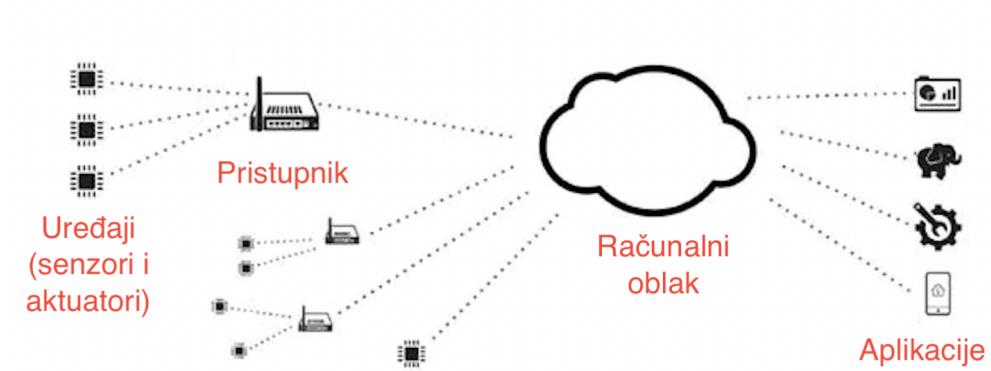
Njihova je funkcionalnost opažanje informacija iz okoline te interakcija s njome. Senzori služe za prikupljanje podataka, a aktuatori omogućuju da sustav obavlja određene radnje. U pametnom IoT sustavu, senzor prikuplja podatke i šalje ih kontrolnom centru. Kontrolni centar zatim ovisno o načinu na koji je programiran koristi te podatke prije nego što instruira aktuatora da izvrše određene radnje [8]. Protok podataka od senzora do aktuatora prikazan je na slici (Slika 2.2).



Slika 2.2: Protok podataka od senzora do aktuatora [30]

2.Povezivost

Senzori i aktuatori povezani su tzv. pristupnikom (engl. gateway). Njegova je uloga komunikacija sa obližnjim senzorima i aktuatorima te prevođenje poruke u standardni format koji se zatim prenosi u uslugu u oblaku na internetu. Komunikacija između pristupnika te senzora i aktuatora provodi se većinom bežično, no moguća je i žičana usluga. Prikaz povezivosti ilustrira slika (Slika 2.3).



Slika 2.3: Povezivost [31]

3. Računalni oblak

Ovaj termin se odnosi na mrežu računala na internetu, što je prikazano na slici (Slika 2.4). Svrha računalnog oblaka je pohrana i analiza podataka radi donošenja parametnih odluka. Analiza podataka se može odnositi na jednostavne odluke, ali i složene algoritme umjetne inteligencije. Korištenjem oblaka korisnici iznajmljuju infrastrukturu, dakle ne kupuju je, te dobivaju točno one resurse koje su tražili. Time se postiže da troškovi nisu veliki, a kapitalne investicije mogu čak biti jednake nuli čime oblak može unaprijediti neku organizaciju. Neke od poznatijih usluga baziranih na oblaku Facebook, Gmail i Twitter [7].



Slika 2.4: Računalni oblak [33]

4. Interakcija čovjeka i stroja

Interakcija čovjeka i stroja (eng. Human-Machine Interaction, HMI) funkcioniра na način da korisnici nadgledaju podatke i analizu na korisničkim sučeljima (engl. User Interfaces, UI) ili pametnim telefonima koji sadrže specifične aplikacije. Time se postiže informiranje korisnika te im se omogućava poništavanje automatskih odluka po potrebi. Primjer interakcije čovjeka i stroja prikazan je na slici (Slika 2.5).



Slika 2.5: Interakcija čovjeka i stroja [32]

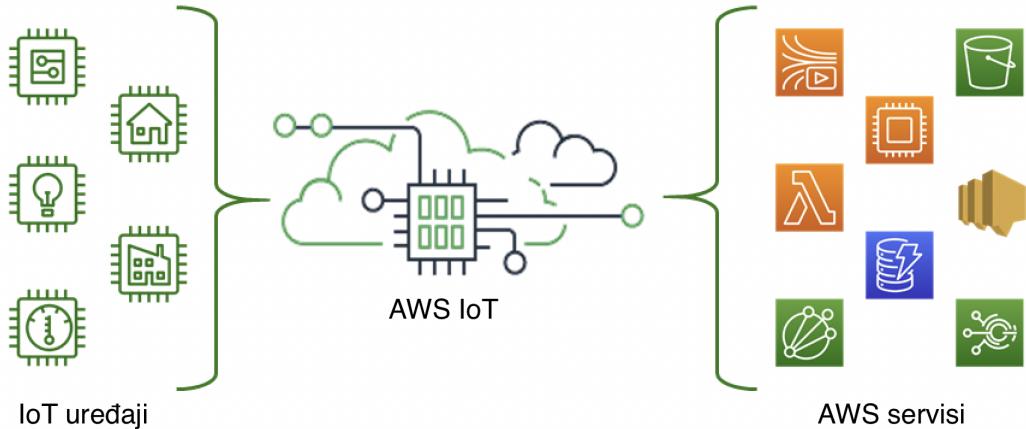
2.2. Pregled postojećih IOT platformi

Postoje mnoge različite IoT platforme dostupne na tržištu koje pružaju infrastrukturu i alate za razvoj, upravljanje i integraciju IoT rješenja. U ovom pododjeljku pružit ću pregled nekoliko popularnih IoT platformi i njihove značajke [9].

1. Amazon Web Services IoT Platforma (AWS IoT)

AWS IoT [10] je izuzetno skalabilna platforma koja se ističe svojom sposobnošću da podržava milijarde uređaja i trilijune interakcija između njih. Svakom interakcijom u AWS IoT-u se može smatrati poruka koja se šalje između uređaja i poslužitelja. Ova platforma omogućuje prikupljanje, pohranu i analizu podataka vezanih za IoT (Internet stvari) u različite svrhe, uključujući industrijske, potrošačke, komercijalne i druge sektore. Prikaz AWS IoT platforme može se vidjeti na slici (Slika 2.6).

Kako bi podržala razvoj aplikacija na AWS-u, AWS IoT pruža razvojni komplet (SDK) koji olakšava programerima izgradnju i pokretanje aplikacija. Ovaj razvojni komplet pruža alate, biblioteke i upute za razvoj aplikacija koje koriste funkcionalnosti AWS IoT-a. Programeri mogu iskoristiti ove resurse kako bi brže i jednostavnije implementirali funkcionalnosti povezane s IoT-om u svoje aplikacije.



Slika 2.6: AWS IoT [34]

2. Microsoft Azure IoT Hub

Microsoft Azure IoT [11, 12] Hub je platforma koja omogućuje povezivanje, upravljanje i nadzor velikog broja IoT uređaja. Pruža sigurnu i pouzdanu komunikaciju između aplikacije za Internet stvari (IoT) i uređaja koje ta aplikacija upravlja. Osim toga, platforma podržava različite protokole te integraciju s drugim Azure uslugama. To znači da korisnici imaju mogućnost prilagoditi i proširiti svoje IoT rješenje prema svojim potrebama.

3. IBM Watson IoT Cloud Platforma

IBM Watson IoT Cloud [13] Platforma pokušava učiniti svoje usluge u oblaku što dostupnijima početnicima s jednostavnim aplikacijama i sučeljima. Omogućava sigurno slanje podatke do oblaka koristeći MQTT protokol za razmjenu poruka. Osim toga, uređaj se mogu postaviti i upravljati putem online upravljačke ploče ili sigurnih API-ja, tako da aplikacije mogu pristupiti i koristiti trenutne i povijesne podatke. IBM Watson također nudi neke zanimljive sigurnosne značajke temeljene na strojnom učenju i znanosti o podacima u svrhu automatizacije poslovnih procesa.

4. Google IoT Cloud Platforma

Googleova IoT Cloud Platforma[14, 15] usredotočuje se na olakšavanje i ubrzanje poslovanja, pružajući mogućnost trenutne obrade podataka u stvarnom vremenu. Ova platforma nudi skalabilnost, visoku razinu sigurnosti te integraciju s drugim Google Cloud uslugama poput BigQuerya i Machine Learninga. Dodatno, korištenje ove platforme omogućuje iskorištavanje privatne globalne optičke mreže Googlea, što pruža pouzdanu i brzu komunikaciju između uređaja i oblaka. Slično kao i Microsoft, Go-

ogle također razvija vlastiti operacijski sustav temeljen na Androidu, što omogućuje jednostavno integriranje i upravljanje IoT uređajima.

5. Thingsboard

Thingsboard [27] je open-source IoT platformu koja omogućava brzi razvoj, upravljanje i skaliranje IoT projekata. Osim toga, pruža podršku za IoT aplikacije, omogućujući povezivanje uređaja, prikupljanje i vizualizaciju podataka te upravljanje njima putem bogatog skupa sučelja. Platforma podržava implementaciju u oblaku i lokalno, pružajući fleksibilnost i prilagodljivost prema potrebama korisnika.

Jedna od ključnih značajki ThingsBoarda je sposobnost konfiguriranja uređaja, resursa i korisnika te definiranje njihovih veza. Ovo omogućuje efikasno organiziranje i upravljanje IoT infrastrukturom.

Prikupljanje podataka i vizualizacija su također ključni aspekti ThingsBoarda. Korisnici mogu daljinski nadzirati uređaje, analizirati podatke i dobiti uvide koji im pomažu u donošenju odluka.

Još jedna važna značajka je podrška za kompleksno upravljanje alarmima putem složene obrade događaja. ThingsBoard omogućava korisnicima definiranje pravila i uvjeta za detekciju odstupanja i generiranje odgovarajućih alarma. Ovi alarmi mogu biti povezani s različitim entitetima u IoT aplikaciji, pružajući detaljna upozorenja o problemima ili nedostacima.

Uz sve ove mogućnosti, ThingsBoard pruža i podršku za daljinsko upravljanje uređajima putem udaljenih procedura poziva (RPC). Korisnici mogu kontrolirati svoje uređaje, izvoditi akcije i konfigurirati postavke putem jednostavnih RPC zahtjeva.

6. Home Assistant

Home Assistant [28] je besplatan softver otvorenog koda za automatizaciju kućanstva. On je centralni kontrolni sustav za pametne kućne uređaje, fokusiran na lokalnu kontrolu i privatnost. Može se koristiti putem web sučelja ili putem aplikacija za Android i iOS, te podržava glasovne naredbe putem virtualnih asistenata poput Google Assistant i Amazon Alexe.

Nakon instalacije Home Assistanta, on djeluje kao centralni upravljački sustav za automatizaciju kućanstva, poznat kao "smart home hub". Omogućuje upravljanje IoT uređajima, softverom, aplikacijama i uslugama putem modularnih integracijskih komponenti. Podržava bežične komunikacijske protokole poput Bluetootha, Zigbee-a i Z-Wave-a, te omogućuje kontrolu otvorenih i vlasničkih ekosustava putem javnih API-ja.

Sve informacije o uređajima i njihovim atributima koje Home Assistant prepoznaće mogu se koristiti i upravljati putem skripti za automatizaciju. Na primjer, može se upravljati osvjetljenjem, klimatizacijom, zabavnim sustavima i kućanskim aparatima.

Navedene platforme samo su neki primjeri popularnih IoT platformi koje pružaju različite mogućnosti za razvoj i upravljanje IoT rješenjima. Postoje i druge platforme kao što su Oracle IoT, Salesforce IoT, Bosch, Cisco IoT Cloud Connect te mnoge druge koje pružaju slične značajke i funkcionalnosti. Ovisno o potrebama i zahtjevima projekta potrebno je odabrat i prikladnu platformu.

2.3. Pregled alarmnih sustava u IoT okruženju

Alarmni sustavi igraju ključnu ulogu u IoT infrastrukturi osiguravajući njenu pouzdanost, sigurnost te optimalnost. U ovome odlomku završnog rada pružit ću nekoliko tipova alarmnih sustava koji se mogu koristiti u IoT okruženju.

1. Detekcija neispravnosti uređaja

Različiti su razlozi neispravnosti uređaja u IoT mrežama - istrošenost baterije, hardverska ili softverska pogreška te mnogi drugi. Alarmni sustavi se mogu ugraditi na uređaje te pratiti njihovo stanje. U slučaju neispravnosti generirati će se alarm što će biti znak korisniku da je uređaj u kvaru i da poduzme odgovarajuće korake kako bi se kvar otklonio.

2. Sigurnosni alarmi

Sigurnosni alarmi su sustavi dizajnirani za otkrivanje neovlaštenih ulaza, poput provale, u zgrade ili druge prostore kao što su domovi ili škole. Njihova svrha je zaštita od provala (krađe) i oštećenja imovine te sprečavanje uljeza. Postoje različiti tipovi sigurnosnih sustava, neki od njih služe samo zaštiti od provala, dok drugi kombiniraju zaštitu od požara i provale. Sustavi za otkrivanje provale često se integriraju s CCTV nadzornim sustavima kako bi se snimile aktivnosti uljeza, a također su povezani s kontrolnim sustavima pristupa koji upravljuju električno zaključanim vratima.

Postoje različite vrste sigurnosnih sustava, a vlasnici kuća često koriste male, samostalne uređaje za alarmiranje. Međutim, postoje i složeniji, višenamjenski sustavi s računalnim praćenjem i upravljanjem. Neki od njih čak uključuju dvosmjerni glasovni sustav koji omogućuje komunikaciju između panela i nadzorne postaje, pružajući dodatnu sigurnost i mogućnost interakcije s korisnicima [16].

3. Detekcija i upozorenje na gubitak veze

Problemi s povezivošću IoT uređaja predstavljaju ozbiljan izazov, jer postoji velik broj potencijalnih točaka kvara. Aplikacijska logika, fizičke mreže, protokoli, hardver i ostale usluge u oblaku mogu sve pridonijeti problemima s povezivošću. Stoga je izuzetno važno imati sposobnost detektiranja i preciznog lociranja izvora tih problema.

U tom kontekstu, alarmi su izuzetno korisni, jer generiraju upozorenja koja omogućuju brzu intervenciju u svrhu ponovne uspostave veze ili otklanjanja problema. Upozorenja nam omogućuju da uspješno rješavanje problema i minimizaciju njihova utjecaja na povezivost IoT sustava [17].

4. Detekcija anomalija

Model za detekciju anomalija koristi umjetnu inteligenciju i strojno učenje kako bi mogao definirati normalne podatke te prepoznati što se to kvalificira kao abnormalno. Kada se uspostave temeljne vrijednosti za izgled podataka sustava kada on ispravno funkcionira, sigurnosni tim definira granice koje pokazuju koliko se različita podatkovna točka mora udaljiti od temeljnog stanja da bi se kvalificirala kao izuzetak. Svaki put kad algoritam detektira podatke izvan tih granica, šalje se administratoru alarm o "detektiranoj anomaliji" [18].

Bitno je za napomenuti kako su navedeni samo neki od mnogih alarma koje možemo koristiti u IoT okruženju. Svaki IoT projekt ima specifične zahtjeve i potrebe pa sukladno tome potrebno je izabrati prikladne alarne kako bi se odgovorilo na te zahtjeve.

3. Spring Boot aplikacija za IoT alarmiranje

3.1. Opis Spring Boot frameworka

3.1.1. Uvod u Spring Boot

Java Spring Boot (Spring Boot) je alat koji omogućuje brži i jednostavniji razvoj web-aplikacija i mikroservisa uz pomoć Spring Frameworka - popularnog okvira otvorenog koda za razvoj samostalnih aplikacija visoke razine koja se izvode na Java virtualnom stroju (JVM). Spring Boot olakšava razvoj aplikacija tako jer preuzima veći dio konfiguracije i podešavanja infrastrukture umjesto programera. To znači da se programeri mogu fokusirati na pisanje koda koji implementira poslovnu logiku, umjesto da gube vrijeme na ručnu konfiguraciju.

3.1.2. Karakteristike Spring Boot frameworka

Spring Boot nudi mnoge karakteristike i prednosti čiji je cilj olakšati razvoj aplikacija. Neke od karakteristika su [19, 20]:

1. Razvoj web aplikacija (engl. Web Development)

Spring Boot je vrlo pogodan za razvoj web aplikacija. Omogućuje nam jednostavno stvaranje samostalnog HTTP poslužitelja koristeći ugrađene poslužitelje poput Tomcata, Jettya ili Undertowa. Funkciju brzog pokretanja i izvršavanja aplikacije ima modul pod nazivom spring-boot-starter-web modul.

2. Klasa SpringApplication

To je klasa koja pruža praktičan način za pokretanje Spring aplikacije koja se može pokrenuti iz glavne metode. Aplikacija se može jednostavno pokrenuti pozivajući statičku run() metodu. Kod koji predstavlja implementaciju klase SpringApplication prikazan je na slici (Slika 3.1).

```
@SpringBootApplication
public class AlarmsApplication {
    public static void main(String[] args) {
        SpringApplication.run(AlarmsApplication.class, args);
    }
}
```

Slika 3.1: Klasa SpringApplication

3. Vanjska konfiguracija

Spring Boot omogućuje vanjsku konfiguraciju (eng. Externalized Configuration) kako bismo mogli raditi s istim kodom aplikacije u različitim okruženjima. Izvori vanjske konfiguracije mogu biti različiti, primjerice Java properties datoteke, YAML datoteke, varijable okruženja i argumenti komandne linije. Na slici (Slika 3.2) prikazana je vanjska konfiguracija u Java properties datoteci.

Osim toga, konfiguracijske vrijednosti se mogu injektirati izravno u beanove koristeći anotaciju @Value, što je prikazano na slici (Slika 3.3). Također konfiguracijske vrijednosti se mogu grupirati i vezati na strukturirane objekte koristeći anotaciju @ConfigurationProperties.

```
api.token=W2xpJzb0r2KuTNenYZJWjAMAg8Zh2GP7eqF4P31pcbXFUrQyvb6rj8fmqQGUWp5w8_Tpm11Xrq0WYAPGxRhYJA==
spring.security.oauth2.resourceserver.jwt.issuer-uri=https://iotat.tel.fer.hr:58443/auth/realmes/spring
```

Slika 3.2: Vanjska konfiguracija u Java properties datoteci

```
@Value("${telegram.bot.token}")
private String botToken;
```

Slika 3.3: Injektiranje konfiguracijske vrijednosti u beanove pomoću @Value anotacije

4. Logging

Spring Boot koristi Commons Logging za interno zapisivanje događaja (logging), ali također pruža mogućnost odabira temeljne implementacije logginga. Zadane konfiguracije pružaju podršku za Java Util Logging, Log4j2 i Logback. Logeri su prethodno

konfigurirani za korištenje ispisivanja na konzolu, uz mogućnost opcionalnog ispisivanja u datoteku.

Izlaz, koji je prikazan na slici (Slika 3.4), obuhvaća sljedeće stavke :

Datum i vrijeme: Preciznost na razini milisekundi i lako sortiranje.

Razina logiranja: ERROR, WARN, INFO, DEBUG ili TRACE.

ID procesa.

Separator — koji označava početak stvarnih poruka logiranja.

Ime dretve: Uokvireno uglatim zagradama (može biti skraćeno za izlaz na konzolu).

Ime logera: Obično je to naziv izvorne klase (često skraćen).

Poruka logiranja.

```
2023-05-23T09:59:56.284+02:00  INFO 1072 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-05-23T09:59:56.290+02:00  INFO 1072 --- [ restartedMain] o.apache.catalina.core.StandardService  : Starting service [Tomcat]
2023-05-23T09:59:56.290+02:00  INFO 1072 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.7]
2023-05-23T09:59:56.321+02:00  INFO 1072 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]      : Initializing Spring embedded WebApplicationContext
2023-05-23T09:59:56.322+02:00  INFO 1072 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 685 ms
```

Slika 3.4: Logging

5. Sigurnost (engl. Security)

Spring Boot aplikacije su web aplikacije bazirane na Springu. Stoga, podrazumijevano je da su sigurne s osnovnom autentikacijom na svim HTTP endpointima. Na raspolaganju je bogat skup Endpointa za razvoj sigurne Spring Boot aplikacije.

6. Automatska konfiguracija

Spring Boot pruža značajku automatske konfiguracije što olakšava razvoj temeljen na ovisnim bibliotekama. Primjerice, ako aplikacija koristi JDBC (Java Database Connectivity) API za pristup relacijskoj bazi podataka obično moramo konfigurirati klase s anotacijom @Bean i JdbcTemplate u Spring aplikacijskom kontekstu. Bitno je za napomenuti kako se ovaj postupak može koristiti u raznim aplikacijama koje koriste JDBC sa relacijskom bazom podataka. Spring Boot je odgovoran za ovu konfiguraciju, a programeru se olakšava tako da ne treba ručno pisati određene dijelove koda. Automatska konfiguracija se može primijeniti i na druge biblioteke poput JPA i sigurnosti. Dakle, Spring Boot omogućuje programerima da brzo i jednostavno razviju aplikacije, smanjujući potrebu za ručnim konfiguriranjem i ponovnim korištenjem standardnih postavki.

3.1.3. Glavne komponente Spring Boot frameworka

Glavne komponente Spring Boot frameworka su [21, 22]:

1. Spring Boot Starteri

Spring Boot Starteri kombiniraju grupe zajedničkih ili povezanih ovisnosti u jednu ovisnost. Primjerice ako želimo razviti Spring Web aplikaciju s Tomcat Web poslužiteljem moramo dodati sljedeće minimalne jar ovisnosti u Mavenov pom.xml datoteku ili Gradleov build.gradle datoteku: Jar datoteka Spring core (spring-core-xx.jar), Jar datoteka Spring Web (spring-web-xx.jar), Jar datoteka Spring Web MVC (spring-webmvc-xx.jar), Jar datoteka Servlet (servlet-xx.jar). Želimo li još nešto vezano uz bazu podataka, moramo dodati jar datoteke Spring JDBC, jar datoteke Spring ORM, jar datoteke Spring Transaction itd.

Dakle, trebali bi definirati mnogo ovisnosti u našim konfiguracijskim datotekama što predstavlja vrlo zamoran posao te povećava veličinu konfiguracijskih datoteka. Rješenje tih problema su upravo Spring Boot Starteri.

Zadaća Spring Boot Startera je da kombinira sve povezane jar datoteke u jednu jar datoteku tako da možemo dodati samo ovisnost o jednoj jar datoteci u naše konfiguracijske datoteke. Prema tome, u prethodno navedenom primjeru dovoljno je dodati samo jednu jar datoteku: "spring-boot-starter-web". Nakon toga će Spring Boot Framework automatski preuzeti sve potrebne jar datoteke i dodati ih u klasni put našeg projekta. Prikaz spring-boot-starter-web ovisnost u pom.xml datoteci može se vidjeti na slici (Slika 3.5).

```
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
```

Slika 3.5: Spring-boot-starter-web ovisnost u pom.xml datoteci

2. Spring Boot AutoConfigurator

Glavna odgovornost SpringBooot AutoConfiguratora je smanjivanje Spring Boot aplikacije. Naime, ako želimo izgraditi Spring MVC aplikaciju pomoću Spring IO platforme trebamo definirati mnogo XML konfiguracija. No, ako koristimo Spring Boot Framework, tada nam nije potrebno definirati te XML konfiguracije jer će se Spring Boot AutoConfigurator pobrinuti za to. Primjerice, ako koristimo jar datoteku "spring-boot-starter-web" u konfiguracijskoj datoteci projekta, Spring Boot AutoConfigurator će automatski riješiti prikaze, rješavače prikaza itd. Također, Spring Boot

smanjuje definiranje konfiguracije putem anotacija. Ako koristimo anotaciju @SpringBootApplication na razini klase, tada će Spring Boot AutoConfigurator automatski dodati sve potrebne anotacije u Java Class ByteCode. SpringApplication anotacija kombinira tri važne anotacije (@Configuration, @ComponentScan, @EnableAutoConfiguration) što prikazuje slika (Slika 3.6).



Slika 3.6: SpringApplication anotacija [22]

3. Spring Boot CLI (Command Line Interface)

Spring Boot CLI predstavlja softver koji je razvijen u svrhu testiranja i pokretanja Spring Boot aplikacija iz naredbenog retka (eng. command line). Pri pokretanju aplikacije iz naredbenog retka interno se koriste komponente Spring Boot Starter i Spring Boot AutoConfigurator za rješavanje svih ovisnosti i izvršavanje aplikacije. Pokretanje aplikacije iz naredbenog retka prikazuje slika (Slika 3.7).

```
spring run AlarmsApplication.java
```

Slika 3.7: Naredbe za pokretanje aplikacije iz naredbenog retka

4. Spring Boot Aktuator

Pri pokretanju Spring Boot web aplikacije Spring Boot Aktuator automatski pruža ime poslužitelja (localhost) te zadani broj porta kao "8080". Koristeći krajnju točku (eng. endpoint) "https://localhost:8080/" korisnik može pristupiti toj aplikaciji. Spring Boot Aktuator također ima funkciju pružanja upravljačkih krajnjih točaka (eng. management endpoints), a za njihovo predstavljanje se koriste HTTP metode zahtjeva poput GET i POST.

3.2. Arhitektura aplikacije za IoT alarmiranje

U ovom poglavlju objasniti će arhitekturu aplikacije za IoT alarmiranje. Arhitektura se sastoji od nekoliko ključnih komponenti koje omogućuju povezivanje IoT uređaja, prikupljanje podataka, generiranje i slanje alarma putem Microsoft Teamsa i Telegrama.

1. IoT uređaji

Aplikacija je usmjerenja na povezivanje sa IoT uređajima. IoT uređaji su fizički uređaji opremljeni senzorima koji prikupljaju podatke te ih šalju aplikaciji. Ti senzori mogu mjeriti temperaturu, vlažnost, svjetlost ili bilo koju drugu vrijednost potrebnu za praćenje određenih parametara.

2. Influx baza podataka

Podaci koji dolaze s IoT uređaja se spremaju u Influx bazu podataka. Ova baza omogućuje učinkovito pohranjivanje i upravljanje podacima koji dolaze s velikog broja IoT uređaja.

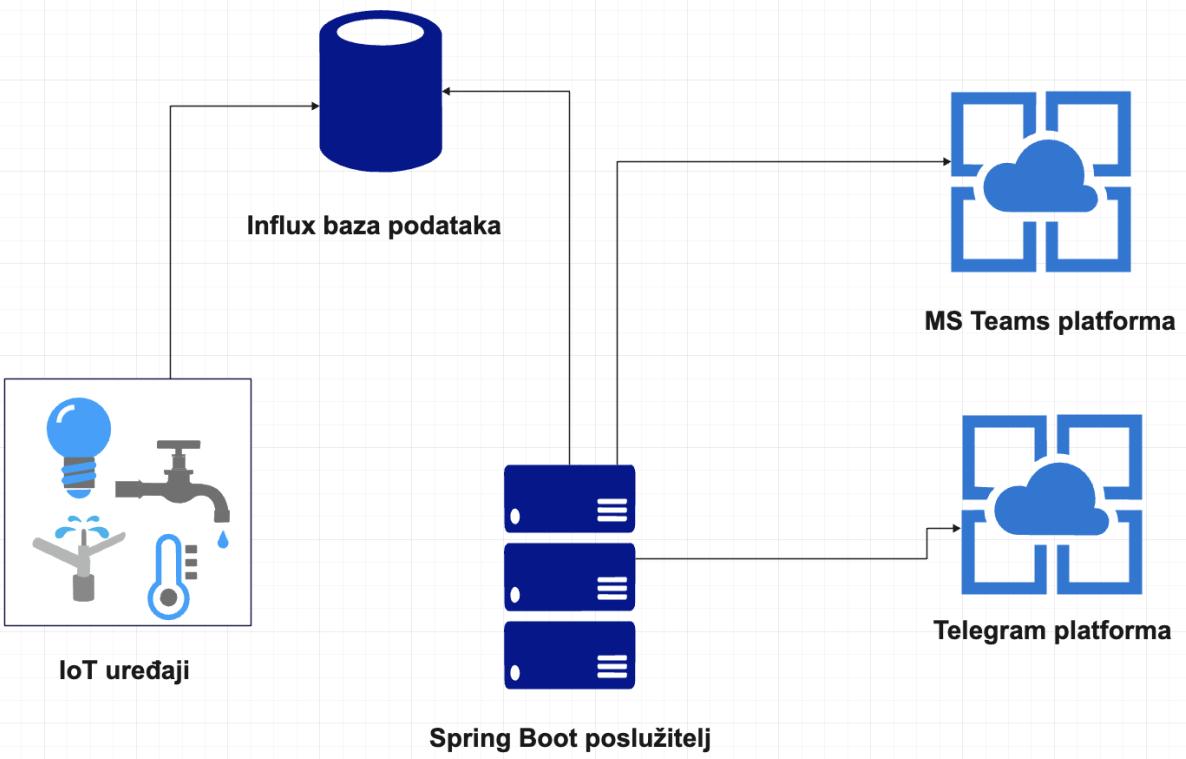
3. Spring Boot poslužitelj

Spring Boot poslužitelj predstavlja srce same aplikacije. On je odgovoran za obradu podataka koji dolaze iz Influx baze, provjeru uvjeta za generiranje alarma i slanje obavijesti korisnicima putem Microsoft Teamsa i Telegrama. Ovaj poslužitelj također pruža REST API koji korisnicima omogućuje postavljanje i upravljanje alarmima te dohvata podataka o senzorima i alarmima.

4. Microsoft Teams i Telegram platforme

Spring Boot poslužitelj koristi Microsoft Teams API i Telegram API kako bi slao generirane alarne korisnicima koji su definirali alarne za praćenje stanja uređaja. Alarne se šalju korisnicima kao obavijesti putem Microsoft Teamsa ili kao poruke putem Telegrama.

Ova arhitektura omogućuje praćenje stanja IoT uređaja, pohranu podataka u InfluxDB bazu, generiranje alarma na temelju definiranih uvjeta te slanje tih alarma korisnicima putem Microsoft Teamsa i Telegrama. Arhitektura aplikacije za IoT alarmiranje može se vidjeti na slici (Slika 3.8).



Slika 3.8: Arhitektura aplikacije za IoT alarmiranje

3.3. Korištene tehnologije i alati

Za implementaciju ove nadogradnje IoT platforme s funkcionalnostima slanja alarma putem Microsoft Teamsa i Telegrama te za testiranje i razvoj koristila sam sljedeće tehnologije i alate:

1. Microsoft Teams

Microsoft Teams [23] se koristio u aplikaciji za slanje generiranih alarma. Predstavlja platformu za komunikaciju koju je razvio Microsoft, kao dio obitelji proizvoda Microsoft 365. Teams je zamjenio druge Microsoftove platforme za poslovnu komunikaciju i suradnju, uključujući Skype for Business i Microsoft Classroom. Omogućava timsku komunikaciju, suradnju te razmjenu poruka, privitaka i obavijesti putem različitih kanala.

2. Telegram

Telegram [24] je još jedan kanal koji se koristio za slanje generiranih alarma. Telegram predstavlja platformu za razmjenu i slanje poruka, obavijesti i primitaka. Aplikacija također pruža opcionalne end-to-end šifrirane razgovore (popularno poznate kao "tajni razgovori") i video pozive i druge značajke. Telegram je prvi put lansiran za iOS 14. kolovoza 2013. godine, a za Android 20. listopada 2013. godine.

3. Eclipse

Eclipse [25] je integrirano razvojno okruženje koje se koristi u računalnom programiranju. Sadrži osnovni radni prostor i proširivi sustav za prilagođavanje okruženja. Trenutno je drugi najpopularniji IDE za Java razvoj, a do 2016. bilo je najpopularniji. Eclipse mi je služio za pisanje koda, upravljanje ovisnostima, debagiranje i testiranje aplikacije.

4. Postman

Postman [26] je alat za testiranje API-ja koji sam koristila za validaciju REST sučelja IoT platforme. Prema podacima iz veljače 2023. godine, Postman izvješćuje da ima više od 25 milijuna registriranih korisnika i 75.000 otvorenih API-ja, što tvrdi da čini najveći javni API centar na svijetu. Postman mi je omogućio slanje različitih HTTP zahtjeva i provjeru odgovora kako bih bila sigurna u ispravno funkcioniranje aplikacije.

4. Generiranje alarma u Spring Boot aplikaciji

4.1. Potreba za generiranjem alarma

Kako bi se osigurala sigurnost te mogućnost pravovremenog reagiranja na probleme s IoT uređajima generiranje alarma u okviru nadogradnje postojeće FER-ove IoT platforme ima ključnu ulogu. S obzirom na to da je svrha IoT uređaja prikupljanje i prijenos podataka putem senzora te izvršavanje naredbi u fizičkom svijetu važno je detektirati kvarove ili uređaje koji nisu u funkciji. Budući da su IoT uređaji konfigurirani tako da periodički šalju podatke, kada se u aplikaciji detektira propušteni termin slanja generirat će se alarm sa prikladnim upozorenjem. Ova nadogradnja omogućuje praćenje stanja IoT uređaja kako bi se mogli identificirati kvarovi na vrijeme, a slanje alarma pomoću Microsoft Teamsa i Telegrama omogućava da odgovorne osobe budu obaviještene na vrijeme o problemima i poteškoćama. Na taj način moći će se poduzeti odgovarajući koraci za rješavanje nepoželjnoga stanja.

4.2. Spring Boot poslužitelj

Poslužitelj sustava je razvijen kao Spring Boot aplikacija. Metoda main se nalazi u razredu `AlarmsApplication`, a sam razred je označen anotacijom `@SpringBootApplication`. Za pokretanje poslužitelja, koristi se metoda run nad razredom `SpringApplication` na sljedeći način:

```
SpringApplication.run(AlarmsApplication.class, args);
```

Nakon što je aplikacija pokrenuta, metoda `sendMessage()` u razredu `DataController`, će periodički provjeravati zapise u bazi podataka, ovisno o vremenskom periodu postavljenom u alarmu. Ako nema odgovarajućih zapisa u bazi, korisniku se šalje poruka o neispravnosti uređaja putem Telegrama ili Microsoft Teamsa,

ovisno o postavkama alarma.

Ovaj mehanizam omogućuje sustavu da automatski detektira kvarove uređaja i obavijesti korisnika o njima putem odabране komunikacijske platforme.

4.3. Detekcija kvara uređaja u aplikaciji

U aplikaciji je implementirana funkcionalnost za detekciju kvara uređaja. Ova funkcionalnost se ostvaruje kroz prethodno spomenutu metodu `sendMessage()`, čiji je kod prikazan na slici (Slika 4.1). Metoda je označena anotacijom `@Scheduled(fixedRate = 10000)`, što znači da će se automatski izvršavati svakih 10 sekundi.

Kada se metoda `sendMessage()` pozove, prvo se iz datoteke čitaju zapisi alarma što je ostvareno u razredu `AlarmFileReader`. Ti zapisi se zatim parsiraju i pretvaraju u objekte `AlarmData`.

Nakon toga slijedi iteracija kroz sve alarme. Za svaki alarm, stvara se instanca `RestTemplate` koja se koristi za slanje HTTP zahtjeva na odgovarajuću URL adresu. U ovom slučaju, URL adresa je "<https://iotat.tel.fer.hr:57786/api/v2/query?org=fer>". Također se postavljaju odgovarajuća zaglavlja za autorizaciju i prihvatanje formata odgovora.

Nakon slanja zahtjeva, provjerava se odgovor. Ako je odgovor prazan, to znači da ne postoji zapis u bazi podataka za vremenski period definiran u alarmu. Tada se, prema postavkama alarma, šalje obavijest korisniku putem odabranе komunikacijske platforme.

Ako je postavljeno da se koristi Telegram za obavještavanje, iz URL-a alarma se izvlače potrebni podaci - token za pristup Telegram botu i ID korisnika. Koristi se objekt `mybot` za postavljanje tokena, a zatim se šalje poruka korisniku pomoću `notificationScheduler-a`.

Ako je postavljeno da se koristi Microsoft Teams, postavlja se URL web kuke preko objekta `teamsconfiguration`, a obavijest se šalje koristeći `teamscontroller`.

Nakon završetka iteracije kroz sve alarme, svi objekti alarma se čiste kako bi se oslobodili resursi.

```

@Scheduled(fixedRate = 10000)
@GetMapping(value="/sendmessage")
public void sendMessage() {

    List<String> alarmEntries = new AlarmFileReader().readDataFromFile(FILE_PATH);
    List<AlarmData> alarms = new AlarmFileReader().parseAlarms(alarmEntries);

    for (AlarmData alarm : alarms) {
        RestTemplate restTemplate = new RestTemplate();
        String url = "https://iotat.tel.fer.hr:57786/api/v2/query?org=fer";
        HttpHeaders headers = new HttpHeaders();
        headers.set("Authorization", "Token " + "W2xpJzb0r2KuTNenYZJWjAMAg8Zh2GP7eqF4P31p"
                + "cbXFUrQyvb6rj8fmqQGUWp5w8_Tpm11Xrq0WYAPGxRhYJA==");
        headers.set("Accept", "application/csv");
        headers.set("Content-type", "application/vnd.flux");

        HttpEntity<String> entity = new HttpEntity<>(alarm.getPayload(), headers);

        ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.POST, entity, String.class);

        if (response.getBody().isBlank()) {
            if(alarm.getAlarmChannel().equals("telegram")){
                int semicolonIndex = alarm.getAlarmUrl().indexOf(';');
                mybot.setBotToken(alarm.getAlarmUrl().substring(0,semicolonIndex));

                String message = alarm.getMessage();
                notificationScheduler.sendNotification(alarm.getAlarmUrl().substring(semicolonIndex+1 ),message);
            } else if(alarm.getAlarmChannel().equals("teams")){
                teamsconfiguration.setWebhookUrl(alarm.getAlarmUrl());
                teamscontroller.sendNotificationScheduled(alarm.getMessage());
            }
        }
        alarms.clear();
    }
}

```

Slika 4.1: Metoda sendMessage

4.4. Postavljanje i konfiguracija alarma u aplikaciji

U ovom poglavlju će se opisati metode koje se koriste za postavljanje i konfiguraciju alarma u aplikaciji.

Metoda addAlarm

Metoda `addAlarm` je definirana kao `@JsonCreator` i `@PostMapping(value="/create")`.

Ova metoda služi za dodavanje novog alarma u aplikaciju. Podaci se primaju u obliku JSON-a i koristi se `ObjectMapper` za pretvorbu JSON-a u objekt tipa `Data`. Nakon uspješne pretvorbe, podaci se zapisuju u datoteku u formatu koji odgovara alarmu. Ukoliko se dogodi pogreška prilikom pretvorbe ili zapisivanja podataka, vraća se odgovarajući HTTP status. Kod koji prikazuje implementaciju metode `addAlarm` prikazan je na slici (Slika 4.2).

Metoda updateAlarm

Metoda `updateAlarm` je definirana kao `@PutMapping("/update/alarmId")` i koristi se za ažuriranje postojećeg alarma. Metoda prima identifikator alarma i novi objekt `Data` koji sadrži ažurirane podatke. Algoritam pretražuje datoteku alarma i pro-nalazi odgovarajući alarm prema identifikatoru. Nakon pronađaska, podaci se ažuriraju

i zapisuju u datoteku. Ukoliko alarm nije pronađen, vraća se status "Not Found". U slučaju pogreške prilikom čitanja ili pisanja datoteke, vraća se odgovarajući HTTP status. Kod koji prikazuje implementaciju metode `updateAlarm` prikazan je na slici (Slika 4.3).

Metoda deleteAlarm

Metoda `deleteAlarm` je definirana kao `@DeleteMapping("/delete/alarmId")` i koristi se za brisanje postojećeg alarma. Metoda prima identifikator alarma i pretražuje datoteku alarma kako bi pronašla odgovarajući alarm za brisanje. Nakon pronalaska, alarm se uklanja iz datoteke. Ukoliko alarm nije pronađen, vraća se status "Not Found". U slučaju pogreške prilikom čitanja ili pisanja datoteke, vraća se odgovarajući HTTP status. Kod koji prikazuje implementaciju metode `deleteAlarm` prikazan je na slici (Slika 4.4).

```
@JsonCreator  
@PostMapping(value="/create")  
public ResponseEntity<String> addAlarm(@RequestBody String jsonData) {  
  
    ObjectMapper mapper = new ObjectMapper();  
    Data data;  
    try {  
        data = mapper.readValue(jsonData, Data.class);  
    } catch (JsonProcessingException e) {  
        e.printStackTrace();  
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Invalid input data");  
    }  
  
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_PATH, true))) {  
        writer.write(String.format("###\n%s\n---\n%s\n---\n%s\n%s\n%s\n",  
            data.getAlarmId(), data.getDeviceId(),  
            data.getDataRequest().getPayload(), data.getAlarmTarget(),  
            data.getAlarmMessage(), data.getAlarmUrl(), data.getAlarmChannel()));  
        return ResponseEntity.ok("Data added successfully");  
    } catch (IOException e) {  
        e.printStackTrace();  
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to add data");  
    }  
}
```

Slika 4.2: Metoda addAlarm

```

@PutMapping("/update/{alarmId}")
public ResponseEntity<String> updateAlarm(@PathVariable String alarmId, @RequestBody Data newData) {
    try {
        List<String> lines = Files.readAllLines(Path.of(FILE_PATH));
        boolean found = false;

        for (int i = 0; i < lines.size(); i++) {
            String line = lines.get(i);
            if (line.equals("###")) {
                String currentAlarmId = lines.get(i + 1);
                if (currentAlarmId.equals(alarmId)) {
                    found = true;

                    lines.set(i + 2, newData.getDeviceId());
                    lines.set(i + 3, "___");
                    lines.set(i + 4, newData.getDataRequest().getPayload());
                    lines.set(i + 5, "___");
                    lines.set(i + 6, newData.getAlarmTarget());
                    lines.set(i + 7, newData.getAlarmMessage());
                    lines.set(i + 8, newData.getAlarmUrl());
                    lines.set(i + 9, newData.getAlarmChannel());
                    lines.subList(i + 10, i + 11).clear();
                    lines.subList(i + 11, i + 12).clear();
                    lines.subList(i + 10, i + 11).clear();
                    break;
                }
            }
        }

        if (found) {
            Files.write(Path.of(FILE_PATH), lines);
            return ResponseEntity.ok("Alarm updated successfully");
        } else {
            return ResponseEntity.notFound().build();
        }
    } catch (IOException e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to update alarm");
    }
}

```

Slika 4.3: Metoda updateAlarm

```

@DeleteMapping("/delete/{alarmId}")
public ResponseEntity<String> deleteAlarm(@PathVariable String alarmId) {
    try {
        List<String> lines = Files.readAllLines(Path.of(FILE_PATH));
        boolean found = false;

        for (int i = 0; i < lines.size(); i++) {
            String line = lines.get(i);
            if (line.equals("###")) {
                String currentAlarmId = lines.get(i + 1);
                if (currentAlarmId.equals(alarmId)) {
                    found = true;
                    lines.subList(i, i + 13).clear();

                    break;
                }
            }
        }

        if (found) {
            Files.write(Path.of(FILE_PATH), lines);
            return ResponseEntity.ok("Alarm deleted successfully");
        } else {
            return ResponseEntity.notFound().build();
        }
    } catch (IOException e) {
        e.printStackTrace();
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to delete alarm");
    }
}

```

Slika 4.4: Metoda deleteAlarm

4.5. Integracija s Microsoft Teamsom i Telegramom za slanje alarma

Za uspješnu integraciju s Microsoft Teamsom i Telegramom, najprije je trebalo kreirati posebne "botove" koji će služiti kao posrednici za komunikaciju s tim platformama. Proces kreiranja botova uključuje generiranje tokena (Telegram) koji se koristi kao autentifikacijski ključ za pristup botu te "Webhook URL-a" (Microsoft Teams) koji služi za uspostavljanje komunikacije između aplikacije i Microsoft Teamsa. Na taj način omogućeno je aplikaciji da komunicira s botovima i šalje poruke na Microsoft Teams ili Telegram platforme. Nakon što su botovi uspješno kreirani i dobiveni token i "Webhook URL", sljedeći korak bila je implementacija metoda koje će biti odgovorne za slanje alarma u slučaju kvara ili nepravilnosti na Microsoft Teams i Telegram platforme.

4.5.1. Kreiranje Telegram bota

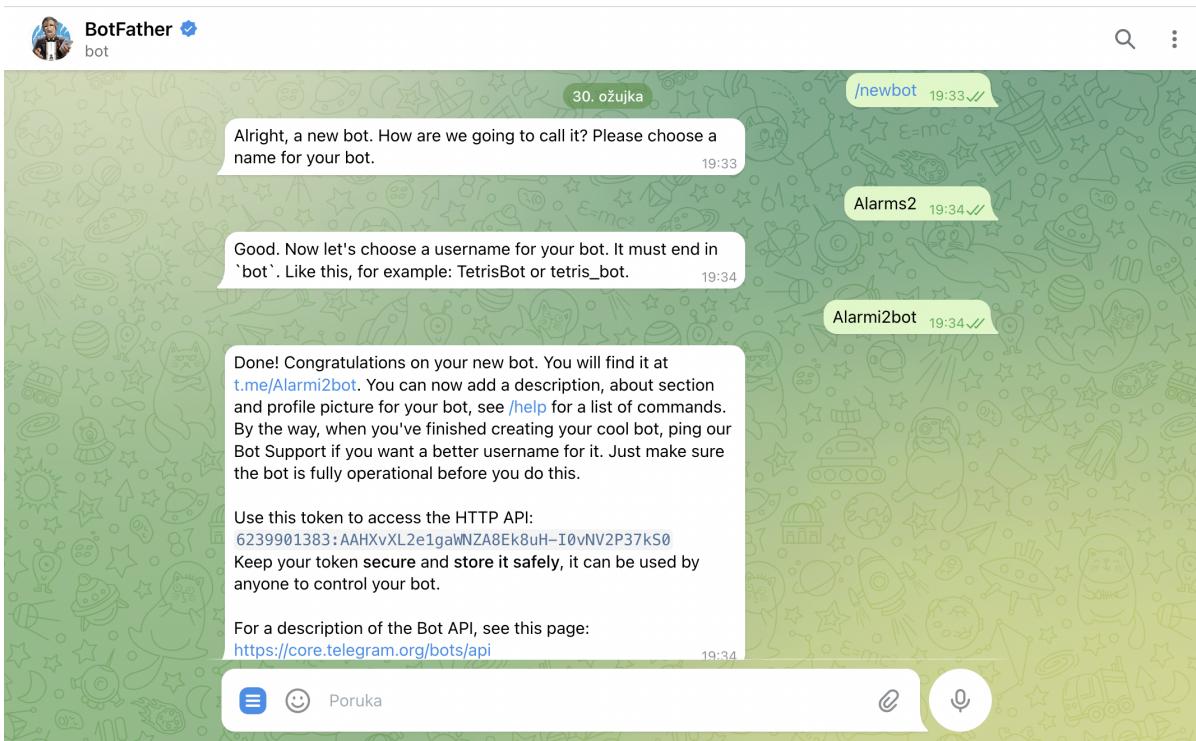
Kreiranje Telegram bota uključuje nekoliko koraka.

Prvi korak je otvaranje razgovora s Telegramovim "BotFatherom". BotFather je službeni bot koji olakšava stvaranje novih botova. U razgovor s BotFatherom treba unijeti naredbu "/newbot" kako bi se započeo proces stvaranja novog bota. Nakon unosa naredbe, BotFather traži da mu se dodijeli ime za novog bota. Ime koje sam odabrala "Alarms2", koristi se za identifikaciju novonastaloga bota.

Sljedeći korak je odabir korisničkog imena (eng. username) za novog bota. Bitno je za napomenuti da korisničko ime treba biti jedinstveno te završavati s "bot". Odabrala sam "Alarms2bot".

Nakon toga BotFather kreira novog bota te šalje poruku s linkom preko kojeg se pristupa botu. Također pruža token koji se koristi za pristup HTTP API-ju bota te neke dodatne upute. Na slici (Slika 4.5) prikazan je proces kreiranja Telegram bota.

Za ostvarivanje komunikacije s botom unutar aplikacije, važno je također imati pridružen Chat ID. Da bismo dobili Chat ID, potrebno je započeti konverzaciju s botom slanjem proizvoljnih poruka. Nakon toga, možemo koristiti web preglednik i posjetiti URL "https://api.telegram.org/bot<bot_token>/getUpdates". Na toj stranici možemo izvući Chat ID koji je povezan s našom konverzacijom. Postupak dobivanja Chat ID-a za ostvarivanje komunikacije s botom unutar aplikacije prikazan je na slici (Slika 4.6).



Slika 4.5: Kreiranje Telegram bota

```
{
  "ok": true,
  "result": [
    {
      "update_id": 582248431,
      "message": {
        "message_id": 2500,
        "from": {
          "id": 6272648058,
          "is_bot": false,
          "first_name": "Valentina",
          "last_name": "Vali\u0107",
          "language_code": "en"
        },
        "chat": {
          "id": 6272648058,
          "first_name": "Valentina",
          "last_name": "Vali\u0107",
          "type": "private",
          "date": 1685443748,
          "text": "kwkw"
        },
        "update_id": 582248432,
        "message": {
          "message_id": 2501,
          "from": {
            "id": 6272648058,
            "is_bot": false,
            "first_name": "Valentina",
            "last_name": "Vali\u0107",
            "language_code": "en"
          },
          "chat": {
            "id": 6272648058,
            "first_name": "Valentina",
            "last_name": "Vali\u0107",
            "type": "private",
            "date": 1685443750,
            "text": "ejj"
          },
          "update_id": 582248433,
          "message": {
            "message_id": 2502,
            "from": {
              "id": 6272648058,
              "is_bot": false,
              "first_name": "Valentina",
              "last_name": "Vali\u0107",
              "language_code": "en"
            },
            "chat": {
              "id": 6272648058,
              "first_name": "Valentina",
              "last_name": "Vali\u0107",
              "type": "private",
              "date": 1685443751,
              "text": "ejj"
            },
            "update_id": 582248434,
            "message": {
              "message_id": 2503,
              "from": {
                "id": 6272648058,
                "is_bot": false,
                "first_name": "Valentina",
                "last_name": "Vali\u0107",
                "language_code": "en"
              },
              "chat": {
                "id": 6272648058,
                "first_name": "Valentina",
                "last_name": "Vali\u0107",
                "type": "private",
                "date": 1685443753,
                "text": "hi"
              },
              "update_id": 582248435,
              "message": {
                "message_id": 2504,
                "from": {
                  "id": 6272648058,
                  "is_bot": false,
                  "first_name": "Valentina",
                  "last_name": "Vali\u0107",
                  "language_code": "en"
                },
                "chat": {
                  "id": 6272648058,
                  "first_name": "Valentina",
                  "last_name": "Vali\u0107",
                  "type": "private",
                  "date": 1685443754,
                  "text": "hi"
                },
                "update_id": 582248436,
                "message": {
                  "message_id": 2505,
                  "from": {
                    "id": 6272648058,
                    "is_bot": false,
                    "first_name": "Valentina",
                    "last_name": "Vali\u0107",
                    "language_code": "en"
                  },
                  "chat": {
                    "id": 6272648058,
                    "first_name": "Valentina",
                    "last_name": "Vali\u0107",
                    "type": "private",
                    "date": 1685443755,
                    "text": "hi"
                  }
                }
              }
            }
          }
        }
      }
    }
  ]
}
```

Chat ID

Slika 4.6: Čitanje Chat ID-a

4.5.2. Implementacija i integracija alarma putem Telegrama u aplikaciji

Implementacija i integracija alarma putem Telegrama u aplikaciji nalazi se u razredima `NotificationScheduler` i `MyBot`.

Razred `NotificationScheduler` je označen anotacijom `@Component` koja ga označava kao komponentu Spring Frameworka. Ovaj razred omogućuje slanje alarma korisnicima. Također, koristi se za komunikaciju s objektom `MyBot`, koji je Telegram bot.

Razred `NotificationScheduler` ima konstruktor koji prima objekt `MyBot` kao ovisnost (dependency). Ovisnost se injektira kroz konstruktor, dakle automatski se

stvara i pruža instanca MyBot razreda prilikom stvaranja NotificationScheduler objekta. To omogućuje NotificationScheduler-u da koristi funkcionalnosti MyBot objekta.

Metoda sendNotification u razredu NotificationScheduler koristi MyBot objekt kako bi slala alarme korisnicima. Prima chatId i poruku postavljenu u alarmu. Slanje poruke odvija se putem telegramBot.sendNotification(chatId, n) te se pritom koristi MyBot objekt. Prikaz razreda NotificationScheduler može se vidjeti na slici (Slika 4.7).

Razred MyBot je također označen anotacijom @Component, što ga čini Spring komponentom. Ovaj razred nasljeđuje TelegramLongPollingBot razred, koji je dio Telegram Bot API-ja i pruža funkcionalnosti za stvaranje i upravljanje Telegram botom. Razred MyBot koji nasljeđuje razred TelegramLongPollingBot prikazan je na slici (Slika 4.8).

Privatno polje botToken sadrži token za pristup Telegram botu. Ovaj token se koristi za autentifikaciju i identifikaciju bota, a novi token se može postaviti pomoću metode setBotToken. U kodu prikazanom na slici (Slika 4.9) možemo vidjeti privatno polje botToken i metodu setBotToken.

Metoda sendNotification prima chatId i tekst poruke, stvara objekt SendMessage s tim podacima te koristi Telegram API kako bi poslala poruku korisniku. Kod prikazan na slici (Slika 4.10) prikazuje metodu sendNotification.

Ova kombinacija razreda omogućuje NotificationScheduleru da koristi funkcionalnosti Telegram bota definirane u razredu MyBot, poput slanja obavijesti korisnicima.

```
@Component  
@EnableScheduling  
public class NotificationScheduler {  
    private final MyBot telegramBot;  
  
    public NotificationScheduler(MyBot telegramBot) {  
        this.telegramBot = telegramBot;  
    }  
  
    public void sendNotification(String chatId, String n) {  
        telegramBot.sendNotification(chatId, n);  
    }  
}
```

Slika 4.7: Razred NotificationScheduler

```
@Component  
public class MyBot extends TelegramLongPollingBot {
```

Slika 4.8: Klasa MyBot nasljeđuje klasu TelegramLongPollingBot

```
    private String botToken;  
  
    public void setBotToken(String token) {  
        botToken = token;  
    }
```

Slika 4.9: Privatno polje botToken i metoda setBotToken

```
    public void sendNotification(String chatId, String messageText) {  
        SendMessage message = new SendMessage();  
        message.setChatId(chatId);  
        message.setText(messageText);  
        try {  
            execute(message);  
        } catch (TelegramApiException e) {  
            e.printStackTrace();  
        }  
    }
```

Slika 4.10: Metoda sendNotification

4.5.3. Kreiranje Microsoft Teams bota

Kreiranje Microsoft Teams bota također se sastoji od više koraka. Najprije sam u timu "IOT Alarms" otvorila novi kanal nazvan "IoTAlarms" u koji želim integrirati bota. Klikom na tri točke pored imena kanala otvorio mi se izbornik s dodatnim opcijama iz kojeg sam odabrala opciju "Connectors". U odjeljku "Connectors" odabrala sam "Incoming Webhook" klikom na nju. Zatim mi se otvorio prozor sa formom u koju je trebalo upisati ime za Webhook, odabrala sam "Alarms", kao što je prikazano na slici (Slika 4.11). Nakon što je Webhook stvoren, prikazao se URL koji mi je potreban za integraciju u aplikaciju. Kopirala sam taj URL jer će mi trebati prilikom kreiranja alarma kako bi se mogla uspostaviti komunikacija sa Microsoft Teamsom. Nakon uspostavljanja konekcije bot će u kanal poslati odgovarajuću poruku o uspostavljenoj konekciji, a prilikom pokretanja aplikacije i generiranja alarma u kanal će se slati odgovarajuća poruka za taj alarm, što se može vidjeti na slici (Slika 4.12).

Connectors for "IoTAlarms" channel in "IoT Alarms" team

X



Incoming Webhook

[Send feedback](#)

The Incoming Webhook connector enables external services to notify you about activities that you want to track. To use this connector, you'll need to create certain settings on the other service, which needs to support a webhook that's compatible with the Office 365 connector format.

Fields marked with * are mandatory

To set up an Incoming Webhook, provide a name and select Create.*

Alarms

Customize the image to associate with the data from this Incoming Webhook.

[Upload Image](#)



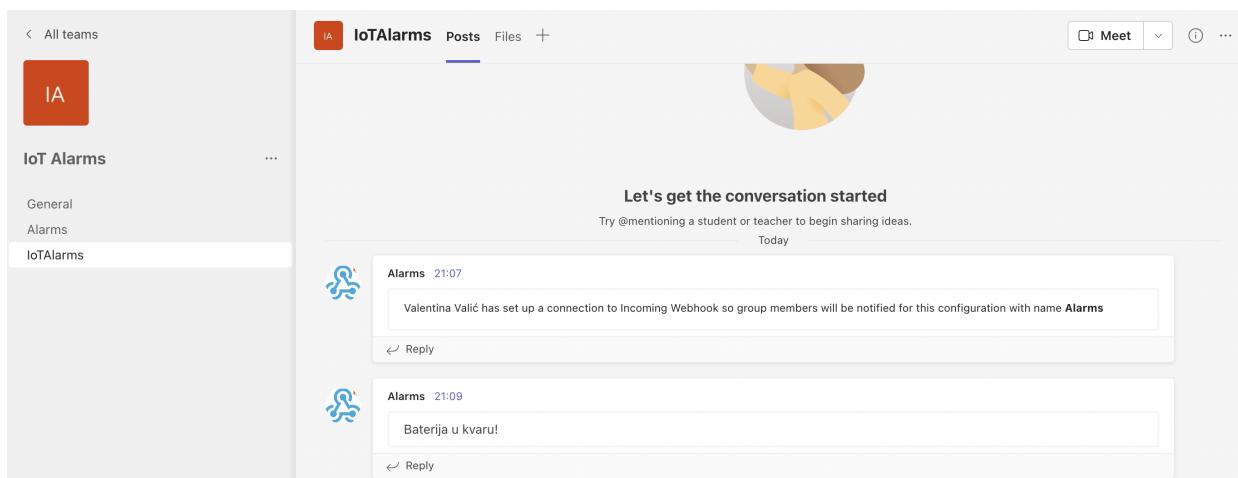
Default Image

[Create](#)

[Cancel](#)

Note: If you're a software developer and want to learn more about sending data to Office 365 using Incoming Webhook, see [Get started with Office 365 Connector Cards](#).

Slika 4.11: Kreiranje Webhook URL-a



Slika 4.12: Microsoft Teams bot

4.5.4. Implementacija i integracija alarma putem Microsoft Teamsa u aplikaciji

Implementacija i integracija alarma putem Microsoft Teamsa u aplikaciji nalazi se u razredima `TeamsConfiguration`, čiji je kod prikazan na slici (Slika 4.13) te `TeamsController`, čiji je kod prikazan na slici (Slika 4.14).

Razred `TeamsConfiguration` predstavlja konfiguraciju integracije s Teams sustavom. U ovom razredu se definira URL za Webhook preko kojeg će se slati alarmi. Metoda `getWebhookUrl()` vraća trenutno postavljeni URL, dok metoda `setWebhookUrl()` omogućava postavljanje novog URL-a.

Također, u ovom razredu se definira `RestTemplate` kao bean koji će se koristiti za slanje HTTP zahtjeva.

Razred `TeamsController` je komponenta koja se koristi za slanje alarma putem Teams sustava. U njemu se koristi `TeamsConfiguration` za dohvrat URL-a Webhooka, te se koristi `RestTemplate` za slanje HTTP POST zahtjeva na taj URL. Metoda `sendNotificationScheduled()` je metoda koja se poziva prema unaprijed definiranom rasporedu (poziva je metoda `sendMessage` u klasi `DataController` koja je označena `@Scheduled` anotacijom) i šalje alarm sa zadanim tekstrom.

```
@Component
public class TeamsConfiguration {
    private String webhookUrl;

    public String getWebhookUrl() {
        return webhookUrl;
    }

    public void setWebhookUrl(String webhookUrl) {
        this.webhookUrl = webhookUrl;
    }

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

Slika 4.13: Razred TemsConfiguration

```
@Component
public class TeamsController {
    @Autowired
    private TeamsConfiguration teamsConfiguration;

    @Autowired
    private RestTemplate restTemplate;

    public void sendNotificationScheduled(String message) {
        String payload = "{\"text\": \"" + message + "\"}";
        restTemplate.postForEntity(teamsConfiguration.getWebhookUrl(), payload, String.class);
    }
}
```

Slika 4.14: Razred TeamsController

5. Primjer korištenja aplikacije za IoT alarmiranje

U ovom poglavlju, opisat ću jedan konkretni scenarij primjene aplikacije. Fokus će biti na slanju REST zahtjeva pomoću alata Postman. Postman mi omogućuje testiranje i interakciju s aplikacijom putem HTTP zahtjeva. Ovo će omogućiti detaljnije razumijevanje rada aplikacije te ispitivanje njezinih funkcionalnosti i performansi.

5.1. Keycloak autentifikacija

Kako bi se uspostavila veza s bazom podataka, prvo je potrebno izvršiti Keycloak autentifikaciju kako bi se izbjegao status 401 Unauthorized. Dakle, šalje se POST zahjev na URL adresu "<https://iotat.tel.fer.hr:58443/auth/realms/spring/protocol/openid-connect/token>". U tijelu zahtjeva potrebno je ispravno postaviti parametre i specificirati korisničko ime i lozinku s kojima želimo ostvariti vezu (u ovom primjeru korisničko ime je postavljeno na "u1", a lozinka također na "u1"). Nakon slanja zahtjeva, u tijelu odgovora dobivamo pristupni token koji se koristi za autorizaciju. Nakon dobiwanja tokena, mogu se slati zahtjevi prema bazi podataka. Primjer Keycloak autentifikacije može se vidjeti na slici (Slika 5.1).

POST https://iotat.tel.fer.hr:58443/auth/realms/spring/protocol/openid-connect/token

Body

Key	Value	Description
grant_type	password	
client_id	rest-keycloak	
client_secret	af252ec6-2d3f-4a26-bbdc-6b1a8785b6b5	
username	u1	
password	u1	

Pretty Raw Preview Visualize JSON

```

1 "access_token": "eyJhbGciOiJSUzI1NiIsInRcCiwiAiSldUiIiwi2lkiIiA6ICiwlXRRPm0yYUVTQXojoDTh3N084WE9RQ0psOFJBNlBwYWVY31feFRaVWxZIn0.
2 eyJleHAiOjE200UzNzIsImhdCi6MTY4NTM3NzM3MiwanRpIjojZTdmYmIzNzNtNmVjNS02Th1lTg2YTMTMnRNTbjOGEOMzdkiIwiaXNzIjoiaHR0cHM6Ly9pb3RhC502Wuu2mVyLmh0YjU4NDQzL2F1d
3 GvcmVnbG1zL3Wcmclu2yIsImI1ZC16ImFjY291bnQilCJzdWIoiIw0QzNTdjNS03MmMSLTRhMzYtYjEzZiiYjBhMjYz0dIdmZaiIc30eXai0iJC2WFyZXIiLCJhenA1o1jyZXN0Lwt1ewNsB2FrIwic2Vzc2
1vb1zQdfGfZS16Imu02TR0iDEylTg0MG1tNDAM108yJ1lC16ImFjIi61jE1lC3hbGxdv2kLw9yaWdpbmM101siAhR0Cbdovl2xvY2fsG9z0d04NTgwIwiaHR0cHM6Ly9pb3RNdC50Z2w
uZnVlyLmh0YjU4NDQzI1OsInJ1YwxtX2FjY2VzcyI6eyJybx2yclyJ6wJmZxJpdCisInZlciIsImRlZmF1bhQtcm9sZXmtc3Byaw5nIwiaW90LX1YjWQ1lCjVznsaW51XZfjY2VzcyIsInVtVY9hdxRo3jPemF0
aW9uIiwiYXbwLXVzX1ixX09sInlc291cnM1X2FjY2VzcyI6eyJhY2Nvd501jptInJvbgVz1jpbm1hbmfnzS1hy2Nvdw501iwiwbFuYwd1LWfjY291bnQtbGlua3M1c12aw3LXbyb2zbpbGUixX19LCjzY29nZ
SI6InByb2ZpbGUgZw1haawiLCJzaWQ1o1JINGU0YjgxMi04NDBiLTwNzIt0WiawZS00NzNkY2MzBhMTEiLCj1bWPfb922XjPzml1ZC16dHJ1ZSwicHJ1ZmVycnVx3VzZxJuY11IjoiidTeiLCjbWfpC161n
UxQGVtYwlsLmVbS1sInByb3R1cmu10i3tb2pIHNsawthIn0.
dd9Cmkts2fHe100T2Q00wpncu1c1H056ro0KjSD7Wmu90FFQ5zKQJaRazoSoGkj_gVA4SmCq-zLm4AK0zvSByoBgtYbr95Tf1K16_z7eU0kzz8U777P2V1v2Teel9oP22NKO3h
Gkn70a0nTi3ge7y6tL1mLcLAPt1EhqvExw2G66xJG31EN3UhHG351tcs1jbg6Ca06_91KmHvn3Bq0Y3t3UR35c0BZpLcAG1j8n0RTgb6zf8P8fjy2R2acaSeG3rQX553jAxNwD6h6cwZ5800st-JcuILXGai
4RVMGHKh1Ujw56Hfpow",
3 "expires_in": 300,

```

Slika 5.1: Slanje POST zahtjeva za Keycloak autentifikaciju i pristupni token u odgovoru

5.2. Dodavanje, ažuriranje i brisanje alarma

Nakon što je aplikacija uspješno pokrenuta te nakon što je uspostavljena autentifikacija, korisnik ima mogućnost interaktivno upravljati alarmima. Moguće je dodavati nove alarne, brisati postojeće i ažurirati već postavljene alarne. U konkretnom primjeru, neka korisnik želi inicialno postaviti četiri alarma za uređaj "SAP01", s odgovarajućim vrijednostima za temperaturu (TC), bateriju (BAT), svjetlost (LW) i vlažnost (HUM).

Nakon ispravnog postavljanja svih parametara u tijelu zahtjeva (Slika 5.2), očekujemo da će se vratiti statusni kod 200 OK, što ukazuje na uspješno dodavanje podataka. U tijelu odgovora, vidjet ćemo poruku "Data added successfully" (Slika 5.3), što potvrđuje da su alarmi uspješno spremjeni u sustavu, tj. zapisani u za njih namjenjenu datoteku (Slika 5.4).

Jedan primjer alarma iz datoteke može biti sljedeći:

Alarm ID: 4028b8818853b8d1018853c187000002

Uređaj: SAP01

Konfiguracija upita za alarm:

FROM: "telegraf"

RANGE: -49h (posljednjih 49 sati)

FILTRIRANJE:

Po mjerenu: "TC"

Po ID-u waspa: "SAP01"

Korisnik: u1

Poruka: Nema dostupnih mjerena za TC!

Konfiguracija za obavještavanje putem Telegrama:

Bot token: 6023056715:AAG6tNbl-9IYUtOfirZHhnjoHstlYU4rwek

Chat ID: 6272648058

Ovaj primjer prikazuje specifične postavke jednog alarma. Identifikacija alarma se vrši putem jedinstvenog ID-a. Uredaj SAP01 je odabran kao ciljni uređaj za ovaj alarm. Konfiguracija upita definira izvor podataka, vremensko razdoblje i filtre za tražene podatke. U ovom slučaju, alarm se aktivira kada nema novih mjerena za temperaturu (TC) na uređaju SAP01. Kada se alarm aktivira, korisnik "u1" će biti obaviješten putem Telegrama koristeći zadani bot token i chat ID.

Nakon što su alarmi zapisani u datoteku, korisniku se automatski šalju obavijesti o alarmima putem Microsoft Teamsa i Telegrama. Ovisno o postavkama koje je korisnik specificirao prilikom dodavanja zahtjeva, određeni alarmi će biti proslijedjeni na Teams kanal, dok će drugi biti poslati putem Telegrama.

Konkretno, korisnik je specificirao da želi primati alarme za "LW" na svom Microsoft Teams kanalu, te za "TC" na Telegramu. Nakon što su alarmi provjereni u odgovarajućem vremenskom razdoblju, korisnik će primiti obavijest na Teams kanalu s porukom da nema dostupnih mjerena za "LW" (Slika 5.5). S druge strane, korisnik će također primiti obavijest na Telegramu s porukom da nema dostupnih mjerena za "TC" (Slika 5.6).

Važno je napomenuti da korisnik nije primio alarne za "BAT" i "HUM" jer je u zahtjevu postavio vremenski period od 78 sati. Stoga, obavijest o nedostatku podataka nije poslana jer još uvijek postoje relevantni podaci u bazi.

none form-data x-www-form-urlencoded raw binary GraphQL JSON ▾

Beaut

```

1  {
2    "alarmId": "4028b8818853b8d1018853c187000002",
3    "deviceId": "SAP01",
4    "extractDataQuery": {
5      "dataFormat": "csv",
6      "timeColumn": "_time",
7      "valueColumn": "_value"
8    },
9    "alarmMessage": "Nema dostupnih mjerena za TC!",
10   "triggerOperator": "<",
11   "triggerValue": "25.5",
12   "checkInterval": "45s",
13   "dataRequest": {
14     "URI": "https://iotat.tel.fer.hr:57786/api/v2/query?org=fer",
15     "method": "POST",
16     "headers": {
17       "Authorization": "a",
18       "Content-type": "application/vnd.flux",
19       "Accept": "application/csv"
20     },
21     "payload": "from(bucket:\"telegraf\")\n  > range(start: -49h)\n  > filter(fn: (r) => r[\"_measurement\"] == \"TC\")\n  > filter(fn: (r) => r[\"id_wasp\"] ==\n    \"SAP01\")"
22   },
23   "alarmTarget": "u1",
24   "alarmChannel": "telegram",
25   "alarmUrl": "6023056715:AAG6tNbl-9IYUtOfirZHhnjoHstlYU4rwek;6272648058"
26 }

```

Slika 5.2: Tijelo zahtjeva za postavljanje alarma

POST ▼ http://localhost:8080/create

Params Authorization Headers (15) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON ▾

1 Data added successfully

body Cookies (1) Headers (14) Test Results

Pretty Raw Preview Visualize JSON 🔗

Status: 200 OK

Slika 5.3: Uspješno kreiran alarm

```

###  

4028b8818853b8d1018853c187000002  

SAP01  

---  

from(bucket:"telegraf")
|> range(start: -49h)
|> filter(fn: (r) => r["_measurement"] == "TC")
|> filter(fn: (r) => r["id_wasp"] == "SAP01")
---  

u1
Nema dostupnih mjerena za TC!
6023056715:AAG6tNbl-9IYUt0firZHhnjoHstlYU4rwek;6272648058
telegram
###  

4028b8818853b8d1018853c187000003  

SAP01  

---  

from(bucket:"telegraf")
|> range(start: -10h)
|> filter(fn: (r) => r["_measurement"] == "BAT")
|> filter(fn: (r) => r["id_wasp"] == "SAP01")
---  

u1
Nema dostupnih mjerena za BAT!
6023056715:AAG6tNbl-9IYUt0firZHhnjoHstlYU4rwek;6272648058
telegram
###  

4028b8818853b8d1018853c187000004  

SAP01  

---  

from(bucket:"telegraf")
|> range(start: -78h)
|> filter(fn: (r) => r["_measurement"] == "HUM")
|> filter(fn: (r) => r["id_wasp"] == "SAP01")
---  

u1
Nema dostupnih mjerena za HUM!
https://ferhr.webhook.office.com/webhookb2/1e8c14ec-a33b-494a-ab60-53390655714d@ca71eddc-teams
###  

4028b8818853b8d1018853c187000005  

SAP01  

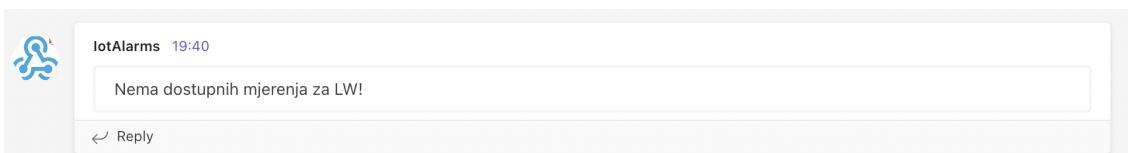
---  

from(bucket:"telegraf")
|> range(start: -6h)
|> filter(fn: (r) => r["_measurement"] == "LW")
|> filter(fn: (r) => r["id_wasp"] == "SAP01")
---  

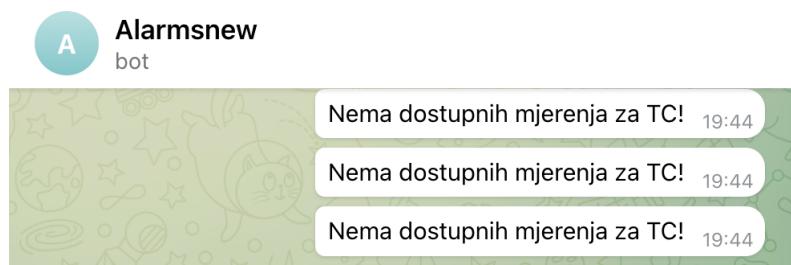
u1
Nema dostupnih mjerena za LW!
https://ferhr.webhook.office.com/webhookb2/1e8c14ec-a33b-494a-ab60-53390655714d@ca71eddc-teams

```

Slika 5.4: Alarmi zapisani u datoteci



Slika 5.5: Obavijest na Teams kanalu s porukom da nema dostupnih mjerena za "LW"



Slika 5.6: Obavijest na Telegramu s porukom da nema dostupnih mjerena za "TC"

Neka korisnik sada želi ažurirati alarm za bateriju (BAT) kako bi se provjeravala mjerena zadnjih 10 sati (Slika 5.7). Da bi to postigao, korisnik šalje PUT zahtjev na endpoint "/update/4028b8818853b8d1018853c18700003", pri čemu broj predstavlja identifikator alarma. Ovaj zahtjev ima za cilj promijeniti postavku intervala provjere na zadnjih 10 sati.

Nakon uspješnog ažuriranja (Slika 5.8), sustav će automatski paralelno s alarmom za temperaturu (TC) slati korisniku i alarm za bateriju (BAT) (Slika 5.9). Razlog tome je nedostatak novih mjerena za bateriju u posljednjih 10 sati u bazi podataka.

```
"payload": "from(bucket:\"telegraf\")\n    |> range(start:-10h)\n    |> filter(fn: (r) => r[\"_measurement\"] == \"BAT\")\n    |> filter(fn: (r) => r[\"id_wasp\"] == \"SAP01\")"
```

Slika 5.7: Promjena payloada u tijelu zahtjeva kako bi se provjeravala mjerena zadnjih 10 sati

Slika 5.8: Uspješno ažuriranje alarma



Slika 5.9: Paralelno slanje alarma za "BAT" i "TC"

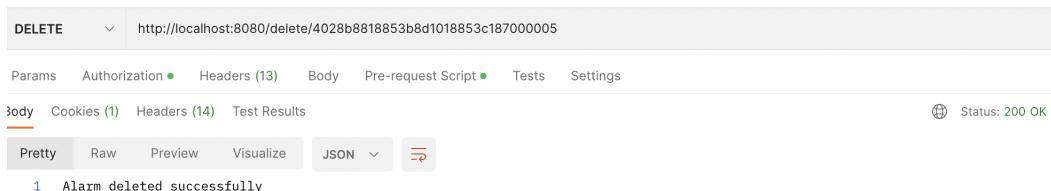
Nakon prethodnih koraka postavljanja i ažuriranja alarma, korisnik sada želi izbrisati određene alarme iz sustava. Konkretno, korisnik želi ukloniti alarme za "LW" i "BAT". Za tu svrhu, korisnik šalje DELETE zahtjeve na endpointove

"`/delete/4028b8818853b8d1018853c187000005`" i

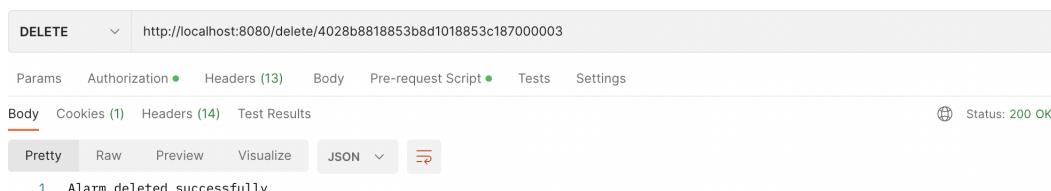
"`/delete/4028b8818853b8d1018853c187000003`",

pri čemu svaki broj predstavlja identifikator pojedinog alarma.

Nakon izvršenja ovih DELETE zahtjeva (Slika 5.10, Slika 5.11), zapisani alarmi za "LW" i "BAT" će biti uspješno uklonjeni iz datoteke (Slika 5.12). U rezultatu, u datoteci će ostati zabilježeni samo alarmi za "TC" i "HUM", koji nisu bili uključeni u brisanje. Nakon uspješnog brisanja alarma za "BAT", korisnik više neće primati obavijesti o bateriji putem alarma. Alarm za "BAT" je uklonjen iz sustava, što znači da korisnik neće dobivati daljnja upozorenja.



Slika 5.10: Uspješan DELETE zahtjev za "LW"



Slika 5.11: Uspješan DELETE zahtjev za "BAT"

```

###  

4028b8818853b8d1018853c187000002  

SAP01  

---  

from(bucket:"telegraf")
|> range(start: -49h)
|> filter(fn: (r) => r["_measurement"] == "TC")
|> filter(fn: (r) => r["id_wasp"] == "SAP01")
---  

u1  

Nema dostupnih mjerena za TC!  

6023056715:AAG6tNbl-9IYUt0firZHhnjoHstlYU4rwek;6272648058  

telegram  

###  

4028b8818853b8d1018853c187000004  

SAP01  

---  

from(bucket:"telegraf")
|> range(start: -78h)
|> filter(fn: (r) => r["_measurement"] == "HUM")
|> filter(fn: (r) => r["id_wasp"] == "SAP01")
---  

u1  

Nema dostupnih mjerena za HUM!
https://ferhr.webhook.office.com/webhookb2/1e8c14ec-a33b-494a-ab60-53390655714d@ca71eddc  

teams

```

Slika 5.12: Alarmi u datoteci nakon brisanja

5.3. Integracija s vanjskim servisima

U okviru ove IoT aplikacije, omogućena je integracija s vanjskim servisima koji korisnicima pružaju dodatne funkcionalnosti i mogućnosti. Kroz tu integraciju, korisnici mogu se pretplatiti na našu aplikaciju te postavljati personalizirane alarne prema svojim potrebama.

Korisnici mogu odabrati različite vanjske servise s kojima žele integrirati našu aplikaciju, kao što su Microsoft Teams, Telegram i slično. Nakon pretplate na aplikaciju, korisnici mogu definirati alarne koji će se aktivirati u slučaju kvara ili neispravnosti uređaja u njihovom IoT sustavu. Na slici (Slika 5.13) prikazan je primjer dodavanja alarma za Microsoft Teams, dok je na slici (Slika 5.14) prikazan primjer dodavanja alarma za Telegram.

Integracija s vanjskim servisima i mogućnost korisničke pretplate na aplikaciju te postavljanje alarma pružaju dodatnu fleksibilnost i personalizaciju korisničkog iskustva, čime se osigurava pouzdan i efikasan nadzor i upravljanje IoT sustavom.

Dodaj novi alarm na uređaj

Test device

Poruka alarma:

Na uređaju nema mjerjenja!

Operator:

>

Vrijednost okidača:

0

Interval provjere:

30s

Kanal:

Microsoft Teams

URL Kanala: [?](https://ferhr.webhook.office.com/webhooks/b2/e494f42d-d4cc-49af-bdaa-d6)

Response extractor

Data format:
csv json

Time column:

Value column:

Slika 5.13: Dodavanje alarma za MS Teams

Dodaj novi alarm na uređaj

Test device

Poruka alarma:

Na uređaju nema mjerjenja!

Operator:

>

Vrijednost okidača:

0

Interval provjere:

30s

Kanal:

Telegram

URL Kanala: [?](6010511055:AAE6Q7gZ21EKT AjkYgHIEaLVi4_RoafWYW4:5876362363)

Response extractor

Data format:
csv json

Time column:

Value column:

Slika 5.14: Dodavanje alarma za Telegram

6. Zaključak

Cilj ovoga rada bilo je istaknuti važnost i potrebu za razvojem i implementacijom IoT platforme za alarmiranje u kontekstu Internet of Things (IoT) tehnologije. Kroz pregled arhitekture i funkcionalnosti platforme, može se primijetiti kako ona omogućuje povezivanje i upravljanje senzorima, prikupljanje podataka, provjeru stanja i generiranje alarma u stvarnom vremenu.

Platforma se sastoji od nekoliko ključnih komponenti, uključujući IoT uređaje, Influx bazu podataka, Spring Boot poslužitelj te integraciju s Microsoft Teams i Telegramom. Kroz njihovu sinergiju, korisnici mogu unaprijed definirati alarne koji se aktiviraju kada senzori ne šalju podatke unutar određenog vremenskog razdoblja. Ti alarni se zatim šalju putem Teamsa i Telegrama, obavještavajući korisnike o mogućim problemima u njihovim IoT sustavima.

Važno je naglasiti da je implementacija ove IoT platforme za alarmiranje vrlo fleksibilna te korisnicima omogućuje prilagodbu alarmnih postavki. Ova platforma pruža korisnicima bolju kontrolu, bržu reakciju na probleme i smanjenje mogućih gubitaka ili kvarova u IoT sustavima.

U zaključku, ova IoT platforma za alarmiranje pruža napredne mogućnosti upravljanja i nadzora IoT sustava, omogućujući korisnicima da budu informirani i djeluju učinkovito u slučaju problema. Njezina implementacija ima potencijal za široku primjenu u različitim industrijama i sektorima gdje je pouzdanost i brza reakcija na kritične događaje od ključne važnosti.

LITERATURA

[1] McKinsey & Company, "What is the Internet of things?", 17.8.2022., [Online]. Dostupno na:

<https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-the-internet-of-things>

[2] "Internet of things", 28.5.2023, [Online]. Dostupno na:

https://en.wikipedia.org/wiki/Internet_of_things

[3] PcChip, "Internet of Things (IoT)", 29.6.2016., [Online]. Dostupno na:

<https://pcchip.hr/internet/internet-things-iot>

[4] MinnaLearn, "Kako IoT funkcionira?", [Online]. Dostupno na:

<https://courses.minnalearn.com/hr/courses/emerging-technologies/the-internet-of-things/how-does-iot-work/>

[5] "Internet stvari", 10.3.2022., [Online]. Dostupno na:

https://hr.wikipedia.org/wiki/Internet_stvari

[6] U.S. Food & Drug Administration, "Radio Frequency Identification (RFID)", 17.9.2018., [Online]. Dostupno na:

<https://www.fda.gov/radiation-emitting-products/electromagnetic-compatibility-emc/radio-frequency-identification-rfid>

[7] Mario Sever, "Računarstvo u oblaku: što je to i čemu služi?", 20.11.2013., [Online]. Dostupno na:

<https://www.ucionica.net/racunala/racunarstvo-u-oblaku-sto-je-to-i-cemu-sluzi-1999/>

[8] Kotai Electronics, "The Ultimate Guide To Sensors And Actuators In IoT", [Online]. Dostupno na:

<https://kotaielectronics.com/ultimate-guide-sensors-and-actuators-in-iot/>

[9] Sam Palmer, "What are the Best IoT Cloud Platforms in 2023?", 2023., [Online]. Dostupno na:

<https://www.devteam.space/blog/what-are-the-best-iot-cloud-platforms/>

[10] Amazon,"AWS IoT Unlock your IoT data and accelerate business growth", [Online]. Dostupno na:

<https://aws.amazon.com/iot/>

[11] Microsoft,"Azure IoT Hub", [Online]. Dostupno na:

<https://azure.microsoft.com/en-us/products/iot-hub>

[12] Microsoft,"Azure IoT Hub concepts overview", 17.3.2023., [Online]. Dostupno na:

<https://learn.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide>

[13] IBM,"About Watson IoT Platform", 5.4.2019., [Online]. Dostupno na:

https://cloud.ibm.com/docs/IoT?topic=IoT-about_iotplatform

[14] ZaranTech,"What makes Google Cloud System different?", 16.11.2022., [Online].

Dostupno na:

<https://zarantech.medium.com/what-makes-google-cloud-system-different-2aa194d4b07a>

[15] "Key Advantages of Google Cloud Platform and Why You Should Adopt It?", 13.10.2020., [Online]. Dostupno na:

<https://www.zarantech.com/blog/key-advantages-of-google-cloud-platform-and-why-you-should-adopt-it/>

[16] "Security Alarm", 10.5.2023., [Online]. Dostupno na:

https://en.wikipedia.org/wiki/Security_alarm

[17]"Monitor, diagnose, and troubleshoot Azure IoT Hub device connectivity", 16.3.2023., [Online]. Dostupno na:

<https://learn.microsoft.com/en-us/azure/iot-hub/iot-hub-troubleshoot-connectivity>

[18] Schuyler Brown, "What Is Anomaly Detection? Methods, Examples, and More", 19.10.2022., [Online]. Dostupno na:

<https://www.strongdm.com/blog/anomaly-detection>

[19] JavaTpoint, "Spring Boot Features", [Online]. Dostupno na:

<https://www.javatpoint.com/spring-boot-features>

[20] Spring, "Core Features", [Online]. Dostupno na:

<https://docs.spring.io/spring-boot/docs/current/reference/html/features.html>

[21] Manoj Debnath, "What Is Spring Boot?", 12.3.2018. [Online]. Dostupno na:

<https://www.developer.com/java/data/what-is-spring-boot/>

[22] Digital Ocean, "Key Components and Internals of Spring Boot Framework", 3.8.2022. [Online]. Dostupno na:

<https://www.digitalocean.com/community/tutorials/key-components-and-internals-of-spring-boot-framework>

[23] "Microsoft Teams", [Online]. Dostupno na:

- https://en.wikipedia.org/wiki/Microsoft_Teams
- [24] "Telegram (software)", 27.5.2023., [Online]. Dostupno na:
[https://en.wikipedia.org/wiki/Telegram_\(software\)](https://en.wikipedia.org/wiki/Telegram_(software))
- [25] "Telegram (software)", 29.4.2023., [Online]. Dostupno na:
[https://en.wikipedia.org/wiki/Eclipse_\(software\)](https://en.wikipedia.org/wiki/Eclipse_(software))
- [26] "Telegram (software)", 6.5.2023., [Online]. Dostupno na:
[https://en.wikipedia.org/wiki/Postman_\(software\)](https://en.wikipedia.org/wiki/Postman_(software))
- [27] "What is ThingsBoard?", [Online]. Dostupno na:
<https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/>
- [28] "Home Assistant", [Online]. Dostupno na:
https://en.wikipedia.org/wiki/Home_Assistant
- [29] "Tech 101: Internet of things", [Online]. Dostupno na:
- [30] Joydeep Misra, "Sensors and Actuators in IoT | Enabling Industrial Automation", 26.6.2017., [Online]. Dostupno na:
<https://bridgera.com/sensors-and-actuators-in-iot/>
- [31] Paul Pickering, "Shaping Smarter Cities: Gateways: The Intermediary Between Sensors and the Cloud", 24.7.2017., [Online]. Dostupno na:
<https://www.mouser.mx/blog/gateways-the-intermediary-between-sensors-and-the-cloud>
- [32] "Human Machine Interface (HMI) for Diagnostic Tool", [Online]. Dostupno na:
<https://www.microchip.com/en-us/solutions/displays/reference-designs-and-application-examples/human-machine-interface-for-diagnostic-tool>
- [33] Luka Crnković, "Primjeri poslovnih primjena računalnog oblaka", 2018. [Online]. Dostupno na:
<https://repozitorij.unipu.hr/islandora/object/unipu%3A2985/dastream/PDF/view>
- [34] Amazon, "What is AWS IoT?", [Online]. Dostupno na:
<https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html>

Usluga u FER-ovoj IoT platformi za alarmiranje u slučaju kvara uređaja

Sažetak

U ovom završnom radu razvijena je IoT platforma za upravljanje alarmima. Platforma omogućuje povezivanje i nadzor IoT uređaja te generiranje alarma u stvarnom vremenu. Implementacija uključuje senzore, bazu podataka, poslužitelj i integraciju s Microsoft Teams i Telegramom. Korisnici mogu postaviti alarne na temelju nedostatka podataka s uređaja. Alarne se šalju putem Teamsa i Telegrama kako bi obavijestili korisnike o problemima u IoT sustavima.

Ključne riječi: IoT platforma, alarne, povezivanje, nadzor, senzori, baza podataka, poslužitelj, integracija, Microsoft Teams, Telegram

Service in FER's IoT platform for device failure alerting

Abstract

This thesis presents the development of an IoT platform for alarm management. The platform enables the connection and monitoring of IoT devices and generates real-time alarms. The implementation includes sensors, a database, a server, and integration with Microsoft Teams and Telegram. Users can set alarms based on the absence of data from devices. Alarms are sent via Teams and Telegram to notify users of issues in their IoT systems.

Keywords: IoT platform, alarms, connection, monitoring, sensors, database, server, integration, Microsoft Teams, Telegram