Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava

Inžinierske dielo

DataHUB

Tím č. 10

Akademický rok: 2021/2022
Predmet: Tímový projekt
Vedúci tímu: Ing. Peter Kaňuch
Členovia tímu: Bc. Erik Podola, Bc. Timotej Lábský, Bc. Viktória Fekete, Bc. Jakub Marinčič, Bc. Matej Glemba, Bc. Volodymyr Otreshko

Kontakt: tpdatahub@gmail.com
Webová stránka predmetu: http://team10-21.studenti.fiit.stuba.sk/

Úvod

IoT svet zažíva už dlhé roky veľký rozmach, ktorý môžeme vidieť všade okolo nás. Na vývoji sa podieľa množstvo inžinierov, nadšencov, študentov vysokých ale aj stredných škôl. Vzniká tak veľké množstvo zariadení, ktoré zbierajú dáta a sú schopné komunikovať s celým svetom pomocou pripojenia na internet. Bohužiaľ pre človeka, ktorý pracuje s IoT ako hobby a nevyvíja veľké aplikácie je veľmi náročné poskytovať namerané dáta do sveta. Preto vidíme veľkú príležitosť vytvoriť jednotnú platformu, ktorá by bola schopná takéto dáta prijímať, perzistovať, vykonávať analýzy a vizualizovať ich pre širokú verejnosť. Mohli by sme tak vytvoriť obrovský zdroj dát pre mnoho vedeckých štúdií, školských alebo hobby projektov a každého, koho by to zaujímalo.

Inžinierske dielo obsahuje všetky potrebné informácie o našom produkte a jeho vzniku. Uvádzame tu cieľe, ktoré sme si stanovili na jednotlivé obdobia vývoja, architektúru nášho systému vrátane nášho protokolu. Popisujeme aj náš pôvodný návrh, následnú implementáciu a výsledky testovania. Nakonci sa nachádzajú praktické príručky.

Globálne ciele

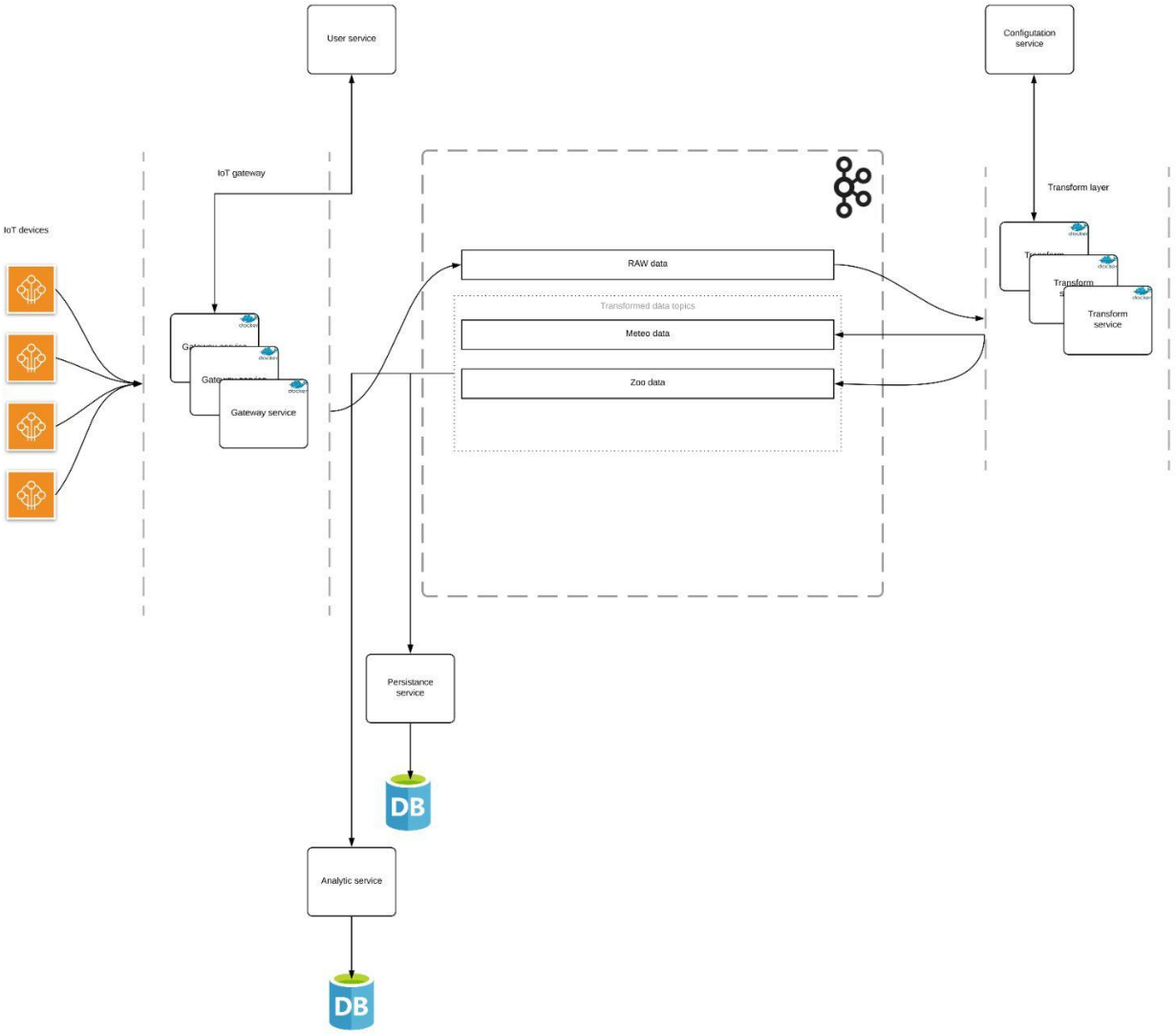
Zimný semester

V rámci tohto zimného semestra sa budeme venovať najmä analýze a návrhu. V analýze bude dôležité oboznámiť sa s problematikou zbierania dát z IoT. Oboznámenie bude zároveň spočívať v preskúmaní navrhovaných a používaných technológií. Následne na základe tejto analýzy bude vytvorený návrh celkového riešenia. Jednotlivé komponenty budú mať formálne definované aplikačné rozhrania a funkčnú špecifikáciu. Následne bude prebiehať implementácia MVP (produkt s najmenšou možnou funkcionalitou - angl.: minimal value product).

Letný semester

Cieľom letného semestra je dopracovať projekt a spraviť testovanie na prvej produkčnej verzií. Začíname s neúplným MVP, takže ako prvé kroky budeme dokončovať všetky moduly z architektúry. Následne vykonáme nasadenie na server, kde vyskúšame pomocou nášho data simulátora flow dát cez všetky služby. Opravíme prípadné nezrovnalosti a nakoniec implementujeme frontend aplikácie. Nakoniec sa pripravíme na testovanie 3. stranou prípravou scenárov a následne vykonáme testovanie.

Architektúra



Obr 1. Architektúra DataHubu

Celý systém

Hlavným cieľom je rozbitie aplikačnej logiky na čo najmenšie časti. Centrálnou časťou je Kafka, ktorej veľkou devízou je horizontálna škálovateľnosť a okolo nej sú vytvorené mikroservisy, aby bola dosiahnutá čo najväčšia nezávislosť.

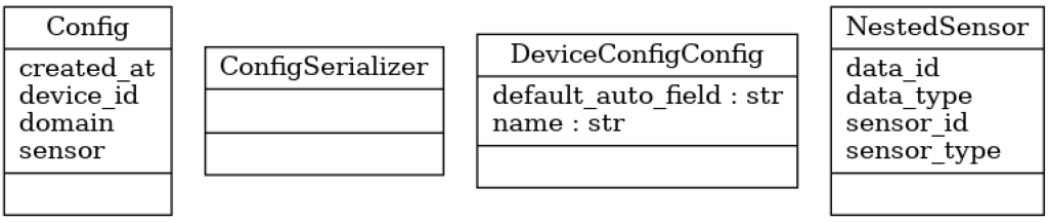
Vstupnú vrstvu tvorí *Gateway service*, ktorá slúži ako server pre náš M2M protokol. *Gateway service* po prijatí dát z IoT zariadenia overí, či konkrétne zariadenie je zaregistrované. Tu vzniká závislosť od *User service*. Po zvalidovaní vstupu, celý payload a metadáta sú publikované ako event do konkrétneho topiku (RAW data). Týmto prístupom vieme plne škálovať vstupnú vrstvu a Kafka garantuje, že o dáta neprídeme ak iné časti systému nebudú v danom čase funkčné.

Kafka, samozrejme, nie je perzistentná vrstva. Náš produkt je schopný na základe nášho návrhu M2M protokolu spracovávať rôzne údaje bez ohľadu na kontext. Na pridanie kontextu do dát slúži transformačná vrstva. Tá je na základe konfigurácie pre konkrétne zariadenie schopná transformovať nespracované dáta (payload) do konkrétnejšej domény. *Transform service* môže mať prípadne rôzne implementácie ak by bola potreba špecifickej implementácie transformácie. Tento prípad nepredpokladáme, že nastane, ale je to možnosť rozšírenia. Transformované dáta sú následne publikované do konkrétnych doménových topickov.

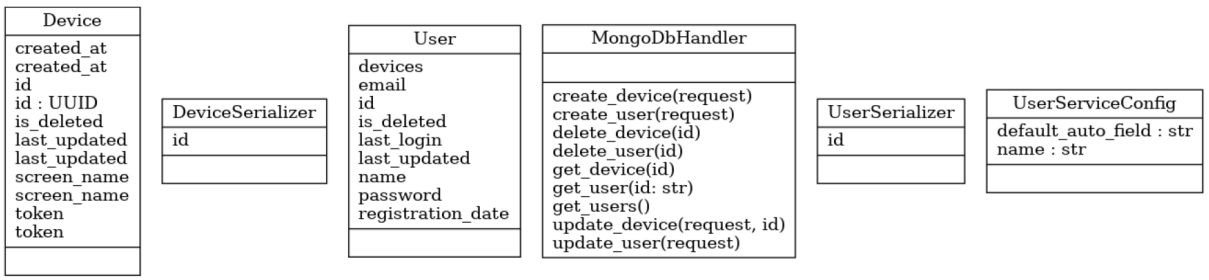
Z konkrétnych doménových dát sa budú ukladať dáta do perzistentnej vrstvy. Až tieto dáta sú prístupné pre vizualizáciu a získavanie cez API.
Architektúra počíta s konkrétnou implementáciou *Analytic service*, ktorá bude doménovo špecifická. Vyhneme sa generickosti, ktorá v tomto prípade môže spôsobiť viacero komplikácií, keďže nemusí byť tak triviálna a konfigurovateľná ako transformačná vrstva. Ako príklad môžeme brať analýzu ovzdušia na základe nameraných hodnôt z meteostaníc. Tieto analyzované dáta budú persistované mimo spracovaných dát.

Celkový návrh sa snaží docieliť nezávislosť a možnosť asynchrónne vykonávať časti, ktoré nie sú na sebe závislé. Týmto spôsobom je možné lepšie využiť hardvérové zdroje kde v určitý čas je dôležité prijímať dáta. Pri znížení záťaže na vstupné dáta bude možné utilizovať hardvér na transformáciu, analýzu alebo iné kroky. Takáto orchestrácia je však závislá na enviromente v ktorom bude táto implementácia bežať. Zároveň umožňuje minimalizovať dopad na používateľov pri vydaní novej verzie len konkrétnej časti produktu.

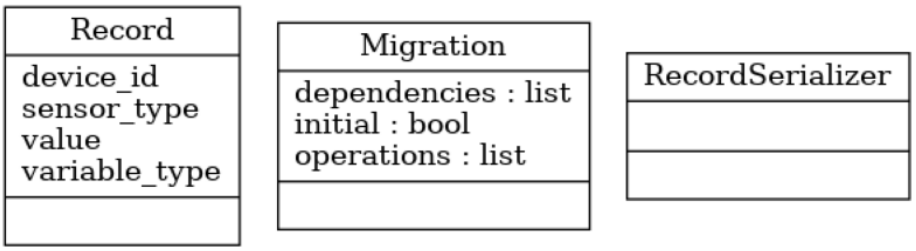
Dátové modely



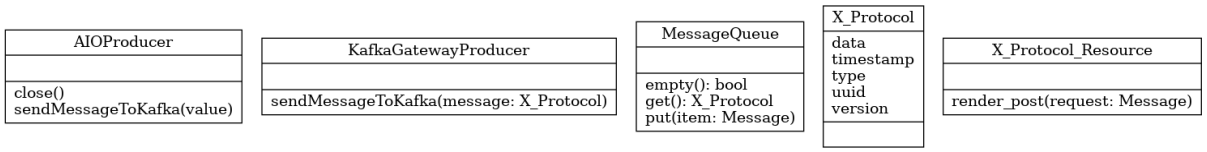
Obr 2. Configuration Service



Obr 3. Device Service

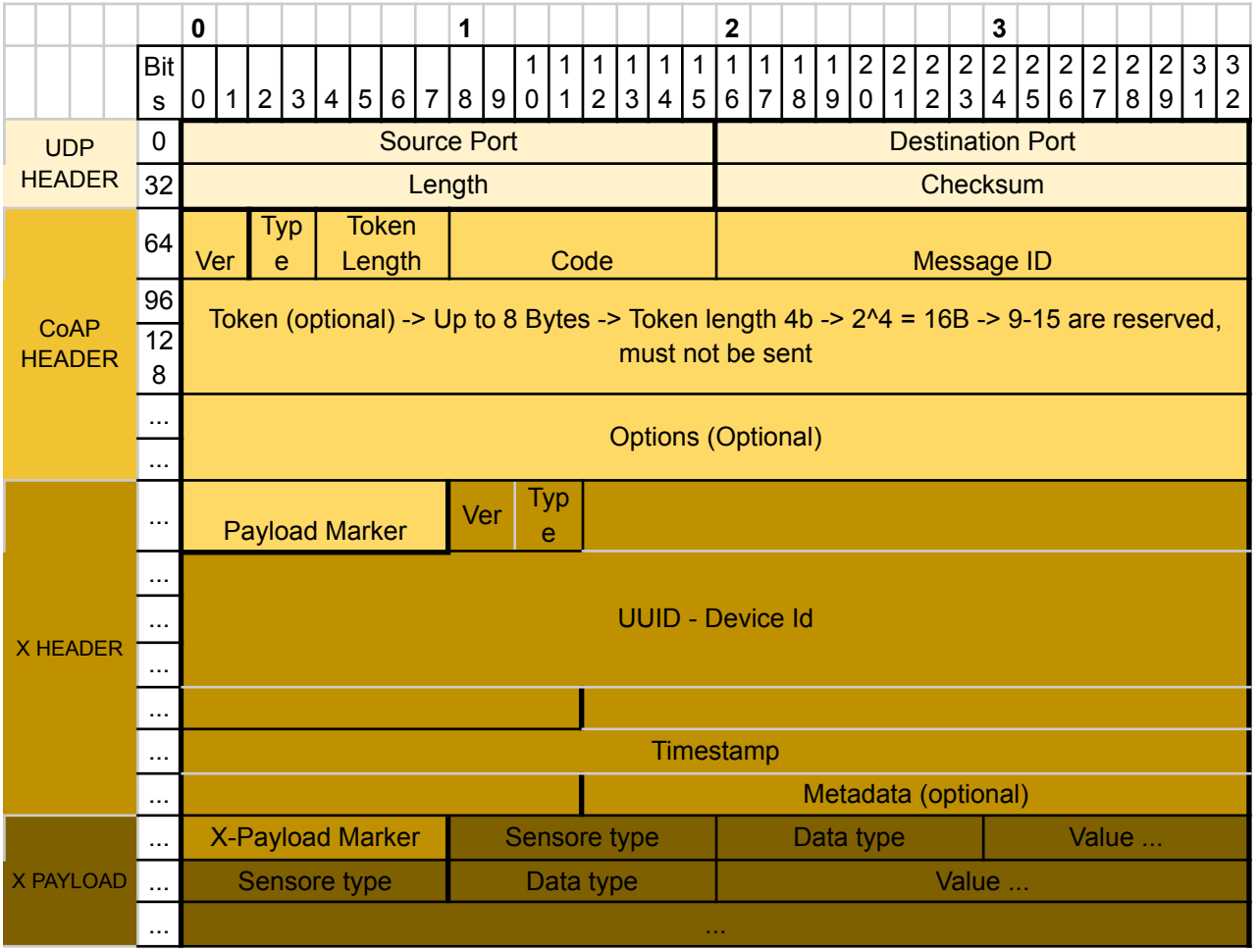


Obr 4. Persistance Service



Obr 5. Gateway Service

XProtocol



Hodnoty

Payload size - length

Veľkosť užitočného zaťaženia protokolu X je možné vypočítať z poľa dĺžky hlavičky UDP, keďže poznáme veľkosť hlavičky CoAP + hlavičky X. Pole dĺžky v hlavičke UDP je povinné.

Checksum
Môžeme použiť kontrolný súčet z hlavičky UDP, keďže tento kontrolný súčet obsahuje hlavičku UDP + údaje UDP (CoAP + X). Pole Checksum je v hlavičke UDP voliteľné, ale vo všeobecnosti sa používa takmer vždy. Tento kontrolný súčet budeme používať iba pre kontrolný mechanizmus. Ak je kontrolný súčet nesprávny, paket jednoducho preskočíme. Typ kontrolného súčtu je 16 bitov CRC.

Version
2 bitové pole, takže môžeme mať 4 verzie X protokolu.

Type
Pole Type označuje typ správy. Ide o 2 bitové pole, takže môžeme mať až 4 typy správ.

- 00 - Data message
- 01 - Metadata message
- 10 - KeepAlive message
- 11 - Unassigned

UUID - device id
Device id je 16B pole pre UUID.

Timestamp
Čas, kedy boli dáta merané. Reprezentácia poľa časovej pečiatky je 64-bitová časová značka Unixu.

CoAP option format = Metadata
Metadáta X budú mať rovnakú štruktúru ako možnosti CoAP

	0								**1**								**2**								**3**							
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	31	31
1	Option delta				Option length				Option value																							
2	Option delta				Option length				Option delta extended								Option length extended								Option value							
3	Option delta				Option length				Option delta extended																Option							
	length extended								Option value																							
	Metadata delta				Metadata length				Metadata value																							

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

Option Format

Bit Positions							
0	1	2	3	4	5	6	7
Option Delta				Option Length			
Option Delta Extended (None, 8 bits, 16 bits)							
Option Length Extended (None, 8 bits, 16 bits)							
Option Value							

Option Delta:

- 0 to 12: For delta between 0 to 12: Represents the exact delta value between the last option ID and the desired option ID, with no Option Delta Extended value
- 13: For delta from 13 to 268: Option Delta Extended is an 8-bit value that represents the Option Delta value minus 13
- 14: For delta from 269 to 65,804: Option Delta Extended is a 16-bit value that represents the Option Delta value minus 269
- 15: Reserved for Payload Marker, where the Option Delta and Option Length are set together as 0xFF.

Option Length:

- 0 to 12: For Option Length between 0 to 12: Represents the exact length value, with no Option Length Extended value
- 13: For Option Length from 13 to 268: Option Length Extended is an 8-bit value that represents the Option Length value minus 13
- 14: For Option Length from 269 to 65,804: Option Length Extended is a 16-bit value that represents the Option Length value minus 269
- 15: Reserved for future use. It is an error if Option Length field is set to 0xFF.

Option Value:

- Size of Option Value field is defined by Option Length value in bytes.
- Semantic and format this field depends on the respective option.

Payload marker : Bajt 0xFF má význam značky užitočného zaťaženia iba tam, kde by sa mohol vyskytnúť začiatok inej možnosti. Značka užitočného zaťaženia je delta opcie a dĺžka opcie s hodnotou 15 -> 1111 1111.

Payload

Sensore type
Typ snímača je 1B pole, takže môžeme podporovať 256 snímačov pre jedno zariadenie

Data type
Typ údajov je pole 1B, takže môžeme podporovať 256 typov údajov.

Value
Údaje zo senzorov. Dĺžka hodnoty je definovaná podľa Data type.

Návrh

Device service

User class model			
Attribute name	**Type**	**Description**	**Required**
id	GUID	Unique ID of user	Yes
name	String	Displayed name of user	Yes
password	String	Secret combination of character for access	Yes
registration_date	Date	Date when user was registered	No
email	String	User's email	Yes
device_ids	Array<GUID>	List of devices ID, which user owns	No
is_deleted	Boolean	Soft delete	No
last_updated	Date	Recent date when any of the attributes was updated	No
last_login	Date	Recent date of login of user	No

.

User service API				
Method's name	**Http type**	**Description**	**Arguments**	**Return type**
create	POST	Creation of new user with at least required parameters	User object	200(Ok), 409(Conflict)
update	PUT	Update of a single attribute or the whole object	User object	200(Ok), 404(Not found)
get	GET	Get user by given ID	user_id	200(Ok), 404(Not found)

delete	DELETE	Soft delete of user	user_id	200(Ok), 404(Not found)

Device class model			
Attribute name	**Type**	**Description**	**Required**
id	GUID	Unique ID of device	Yes
screen_name	String	Displayed name of device	Yes
is_deleted	boolean	Soft delete	No
created_at	Date	Date when user was registered	No
last_updated	boolean	Recent date when any of the attributes was updated	No
token	String	Authorization token of device	No

Device service API				
Method's name	**Http type**	**Description**	**Arguments**	**Return type**
create	POST	Creation of new device with at least required parameters	Device object	200(Ok), 409(Conflict)
update	PUT	Update of a single attribute or the whole object	Device object	200(Ok), 404(Not found)
get	GET	Get device by given ID	device_id	200(Ok), 404(Not found)
delete	DELETE	Soft delete of device	device_id	200(Ok), 404(Not found)
generate_token	POST	Generate authorization token for device	device_id	200(Ok), 404(Not found)
authorization_device	GET	Authorization protocol for device based on	device_id, token	200(Ok), 404(Not found)

		token		

Configuration service

Configuration class model			
Attribute name	**Type**	**Description**	**Required**
id	GUID	Unique ID	Yes
domain	String	Domain name under which the senzor operates (e.g. meteorology)	Yes
senzors	Array<Senzor>	Array of senzors containing	No
created_at	Date		No

.

Sensor class model			
Attribute name	**Type**	**Description**	**Required**
data_type	DataType	Object contains	Yes
senzor_type	String	Type of senzors which will help to categorise them (e.g. Temperature - HIghway)	Yes

DataType class model			
Attribute name	**Type**	**Description**	**Required**
device_id	int	Is the same as Device	Yes
variable_type	String	Enumerated value of supported variables (e.g. int, string)	Yes

Configuration service API				
Method's name	**Http type**	**Description**	**Arguments**	**Return type**
create	POST	Creation of new device configuration with at least required parameters	Configuration object	200(Ok), 409(Conflict)
update	PUT	Update of a single attribute or the whole object	Configuration object	200(Ok), 404(Not found)
get	GET	Get device configuration by given ID	device_id	200(Ok), 404(Not found)
delete	DELETE	Soft delete of device configuration	device_id	200(Ok), 404(Not found)

Kafka udalosti (event)

Event model - IoTPayloadRecieved			
Attribute name	**Type**	**Description**	**Required**
payload	bytearray		Yes
device_id	GUID		Yes
measured_at	timestamp	UNIX timestamp of the measured payload	No

Even model - TransformedData			
Attribute name	**Type**	**Description**	**Required**

sensor_list	List(Sensor)	Array of sensor objects with measured data	Yes
device_id	GUID		Yes
measured_at	timestamp	UNIX timestamp of the measured payload	Yes
Model Sensor			
variable_type	Object.Class	Class type of data	Yes
sensor_type	string	name of senzor	Yes
data	object	measured value	Yes

Implementácia

V rámci “funkčnej” časti riešenia bolo implementované tieto služby:

- dh-frontend
- dh-configuration-service
- dh-device-service
- dh-gateway-service
- dh-transform-service
- dh-persistence

Každá z vyššie uvedených služieb slúži na jednu konkrétnu doménu. Separácia a presná zameranosť pomáha možnosti horizontálneho škálovania na podľa aktuálnej potreby celého systému. Ak služba poskytuje ukladanie dát, má k sebe samostatnú inštanciu nezávislej databázy. Pre ukladanie metadát o zariadeniach a používateľoch sú využité NoSql databázy MongoDB. Ich oddelenie umožnilo paralelný vývoj služieb.

Použité webové frameworky

Niektoré zo služieb využívajú podľa potreby tieto frameworky:

Django

Pre služby vystavujúce RESTful rozhranie sme využili Django framework pre programovací jazyk Python. Django poskytuje ORM vrstvu pre integráciu s databázami.
Služby:

- dh-device-service
- dh-persistence
- dh-configuration-service

ReactJS

Pre používateľské rozhranie je využitý webový framework ReactJS.
Služby:

- dh-frontend

Použité knižnice

V rámci riešení sme využili viaceré voľne dostupné knižnice. Medzi niektoré patria napríklad:

aiocoap

Komunikácia s IoT zariadeniami resp. prijímané dáta sú pomocou našej nadstavby nad Coap protokolom. Pre spracovanie nášho protokolu sme vytvorili wrapper nad *aiocoap* knižnicou.

confluent-kafka

Komunikácia respektíve odosielanie a prihlásenie sa na odber udalostí je implementovaný pomocou *confluent-kafka* v rámci všetkých modulov komunikujúcich s kafkou.

Interné knižnice

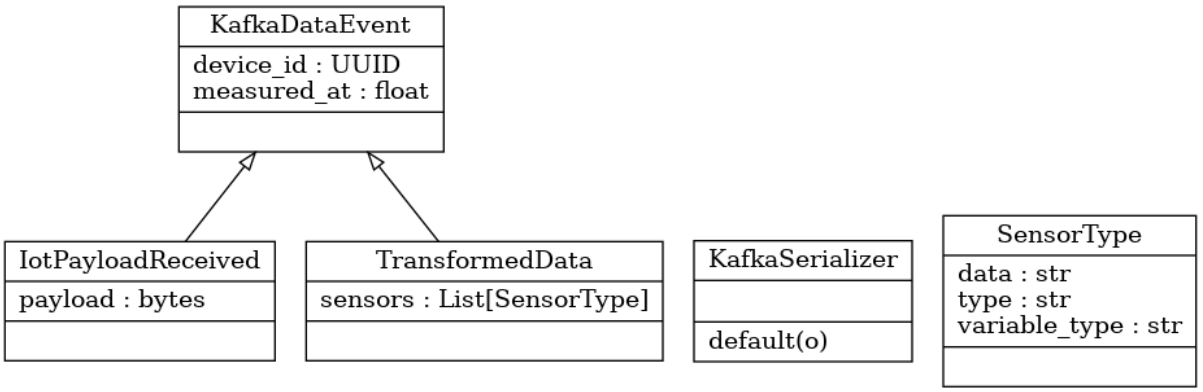
Pre komunikáciu a jednotný model komunikácie sme implementovali jednoduché DTO triedy. Tieto triedy sú prepoužité pomocou **Poetry** manažmentu závislostí.

Dh-kafka

Repozitár dh-kafka obsahuje implementáciu pre jednotnú komunikáciu pomocou Kafka topickov.

V data.py sa nachádzajú tieto triedy s parametrami pre kafka udalosti:

- **KafkaDataEvent** - abstraktná trieda
- **IotPayloadReceived** - data prijaté a spracované gateway službou
- **SensorType** - informácie z konkrétneho sensoru

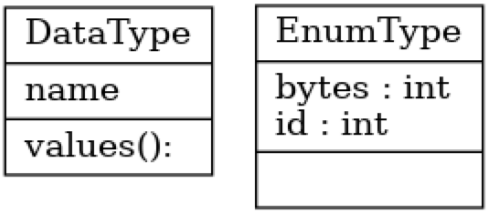


Obr 6. Kafka eventy a serializér

Pre jednotnú serializáciu objektov udalostí je implementovaná trieda KafkaSerializer.

Dh-commons

Pre jednotnú identifikovateľnosť dátových typov je implementovaný enum model **DataType**. Ten obsahuje id a dĺžku payloadu v bajtoch.



Obr 7. Modely, ktoré používa viacero služieb máme vystavené v samostatnom importovatelnom projekte

Github actions

Pre continuous integration sme použili Github Actions. Všetky Docker image sú automaticky buildované a zverejnené pre stiahnutie v **Github Packages.** V niektorých moduloch sa nachádzajú v prípade potreby aj testy, ktoré sú automaticky spustené.

Manažment závislostí

Poetry je nástroj pre manažment závislostí a packaging. Každá implementácia má .toml súbor s definovanými závislostami. Vďaka poetry je možné jednoducho vytvoriť virtuálny environment so všetkými potrebnými závislostami. Pri dokerizácií je taktiež použitá pre získanie potrebných balíkov a knižníc.

Poetry zároveň umožňuje závislostí na github repozitáre. Vďaka tomu je možné manažovať aj vnútorné závislosti.

Testovanie

Náš systém otestovala 3. strana, člen amatérskych meteorológov Jakub Dubec. Test prebiehal v produkčnom prostredí, nasadený na serveri, pričom počas testu bol použitý súkromný notebook testera na prístup do aplikácie. Dáta z IoT zariadenia boli vygenerované priamo počas testovania priloženým simulátorom.
Postupovali sme podľa nasledujúcich testovacích scenárov:

Testovacie scenáre pre desktop

Testovací scenár	1. Registrácia používateľa	**Úspešnosť**
Kroky	1. Zaregistruj sa	☑
	2. Vytvor zariadenie (Name=zariadenie1, Domain=domena1)	☑
	3. Uprav e-mail používateľa (jozko.mrkvicka@gmail.com)	☑
	4. Odhlás sa	☑
Čas mm:ss	00:57	
Poznámky		
Feedback, pocity, návrh na vylepšenie	Pri editácií ma splietli na oko editovateľné textové polia. Čakal by som ich až v momente, keď som už stlačil tlačidlo editovať. Keď máš formulár, kde si pýtaš “Doména”- čo je tá doména? Chýba	

	mi tam vysvetlenie pre ľudí, čo nemajú kontext o aplikácii.

Testovací scenár	2. Uprav nastavenia senzorov	**Úspešnosť**
Kroky	1. Prihlás sa	☑
	2. Zmeň dátový typ jedného senzora	☑
	3. Zmeň typ senzoru jedného zariadenia	☑
	4. Pridaj ďalší senzor pre existujúce zariadenie	☑
	5. Odhlás sa	☑
Čas mm:ss	2:22	
Poznámky		
Feedback, pocity, návrh na vylepšenie	Všetko ok	

Testovací scenár	3. Simulácia dát	**Úspešnosť**
Kroky	1. Prihlás sa	☑
	2. Zobraz prehľad persistovaných dát	☑
	2. Vyfiltruj dáta za posledný týždeň	☑
	3. Koľko je za dané obdobie perzistovaných dát?	☑
	4. Vyfiltruj 2 senzory z predchádzajúceho kroku	☑
	5. Koľko je perzistovaných dát pre vybrané senzory?	☑
Čas mm:ss	1:13	
Poznámky		
Feedback, pocity, návrh na vylepšenie	Som spokojný z mojej strany Žiadny problém, len niektoré drobnosti preferujem inak.	

Testovacie scenáre pre mobil

Testovací scenár	4. Uprav nastavenia senzorov	**Úspešnosť**
Kroky	1. Prihlás sa	☑
	2. Zobraz prehľad zariadení	☑
	3. Zmeň dátový typ senzora “Zariadenie 1”	☑
	4. Odhlás sa	☑
Čas mm:ss	00:57	
Poznámky		
Feedback, pocity, návrh na vylepšenie	Na prvý pokus nereagovalo skrolovanie tabulky (iPhone 12). Na inom zariadení sa nepodarilo problém replikovať.	

Zhodnotenie testovania

Výsledky tímu 10 sú veľmi dobré. V aplikácií chýbajú vysvetlivky významu políčok, no ide len o vizuálnu stránku. Optimalizačne je implementácia na vysokej úrovni. Vzhľadom na charakter projektu sa vysporiadali s rôznymi problémami, ktoré sa počas simulácie vyskytli. Celkovo vieme zhodnotiť túto prácu ako veľmi dobrú. Tím 10 sa zaoberal veľmi zaujímavou a užitočnou témou a dokázal splniť požiadavky zo zadania.

Inštalačná príručka

Prerekvizity:

- Docker engine
- Docker-compose
- SSH kľúč s prístupom k repozitárom (voliteľné pri vytváraní docker obrazov)
- PAT - minimálne práva pre čítanie

Build docker image

(Tento krok je voliteľný - možnosť stiahnuť (pull) obrazy z github packages)

Konvencia pre compose files - dh_gateway, dh_persistence ….

- “-” nahradena “_”

Build images so závislosťami na iné súkromné repozitáre:

Tieto repozitáre je potrebné buildovať týmto spôsobom:

- dh-gateway
- dh-persistence
- dh-transform-service
- dh-data-simulator

Tieto služby obsahujú závislosť na súkromné repozitáre zabezpečené pomocou poetry. Poetry potrebuje mať prístup k repozitárom daných závislostí preto je potrebné aby mala ssh kľúč k týmto repozitárom - argument **BUILD_SSH_KEY**.

cd <<target_dir>>
docker build -t <<image_name>> --build-arg BUILD_SSH_KEY="$(cat path/to/private/key)" .

Build images bez závislostí

Tieto repozitáre je potrebné buildovať týmto spôsobom:

- dh-frontend
- dh-configuration-service
- dh-device-service

cd <<target_dir>>
docker build -t <<image_name>> .

Spustenie pomocou docker-compose

Prerekvizity

Nie ktore služby potrebujú dodatočnú konfiguráciu:

Perzistenčná služba

Perzistenčná vrstva potrebuje vytvoriť alebo nakonfigurovať tieto veci.

Docker volume:

.pg_service.conf - konfigurácia pripojenia služby na databázu

- Príklad obsahu súboru:

  [tsdb_service]
  host=localhost
  user=postgres
  dbname=postgres
  port=5432
  password=password

- Vytvorenie docker volume:

  docker volume create --name pg_service --opt type=none --opt device=/path/to/my/dh/secrets --opt o=bind

Definovanie ENV premenných

- **DH_TSDB_PASSWORD** - heslo pre tsdb postgre používateľa
- **DH_TSDB_TEC_USER_PASSWORD** - heslo pre tsdb tecauser používateľa
- **DH_TSDB_INTERNAL_PASSWORD** - heslo pre komunikáciu medzi AN/DN uzlami ako tecauser používateľ

Kafka

Pre ukladanie dát udalostí je potrebné nakonfigurovať:

Definovanie ENV premenných

- **DH_KAFKA** - cesta pre ukladanie eventov na hoste/storage server.

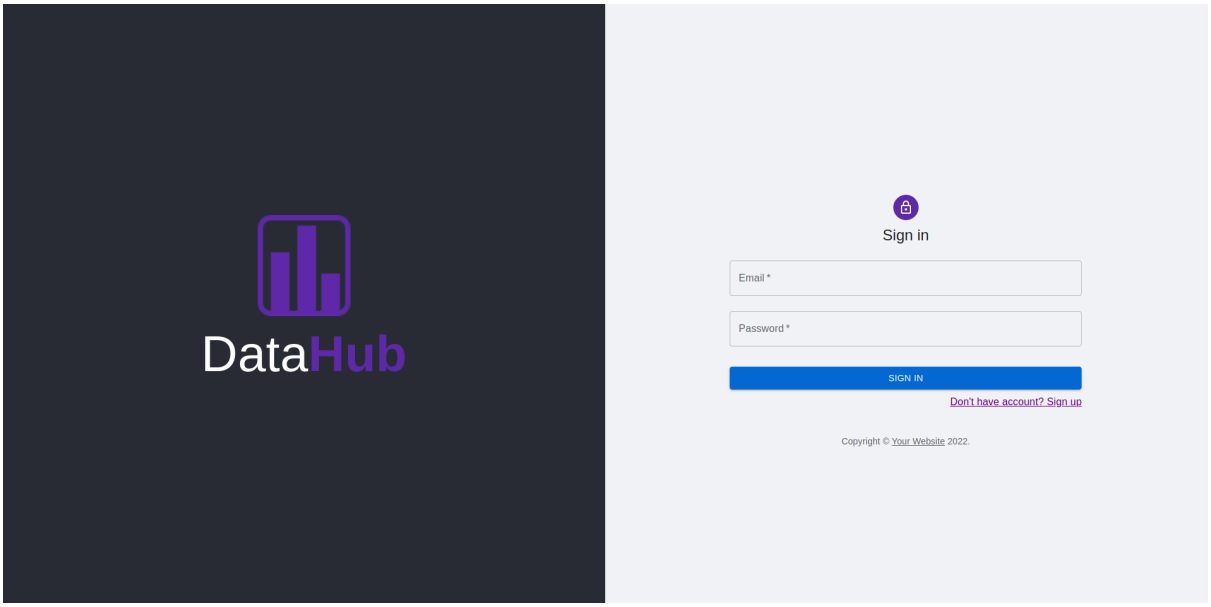
Spustenie docker containerov

Jednotlivé služby majú vlastné docker-compose súbory. Pre kompletné spustenie je možné spustiť pomocou:

docker-compose -p dh -f configurations-service/docker/docker-compose.yml -f device_service/docker/docker-compose.yaml -f gateway/docker/docker-compose.yml -f kafka/docker/docker-compose.yaml -f nginx/docker-compose.yaml -f persistence/docker/docker-compose.yml -f transform/docker-compose.yml -f monitoring/docker-compose.yaml -f grafana/docker-compose.yaml -f frontend/docker-compose.yml up -d

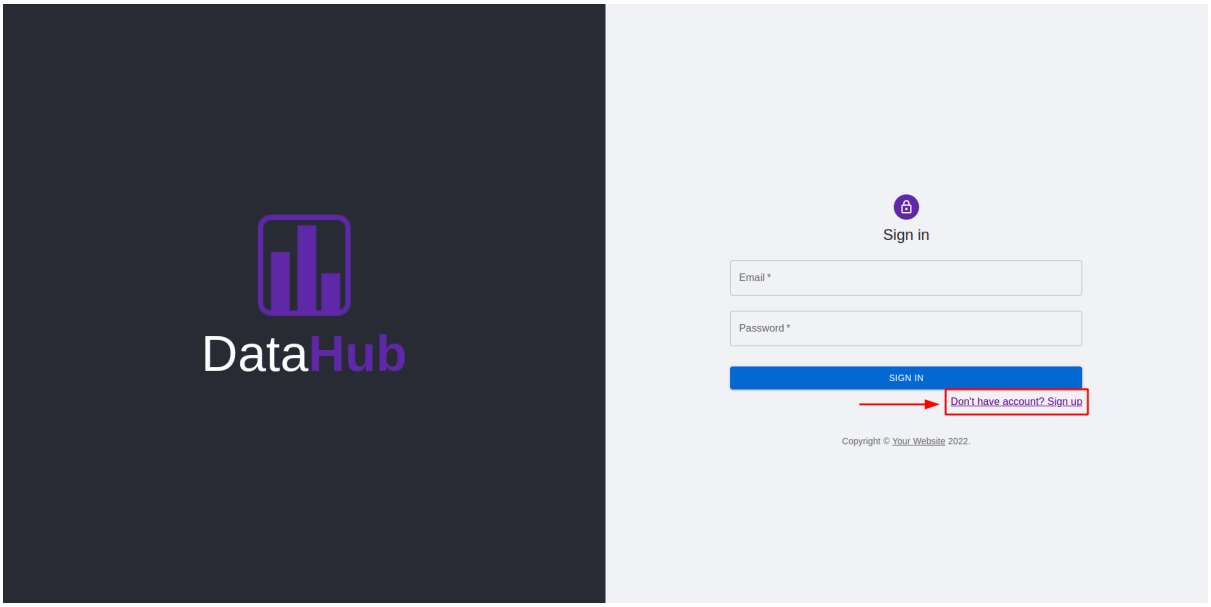
Používateľská príručka

Úvodná obrazovka

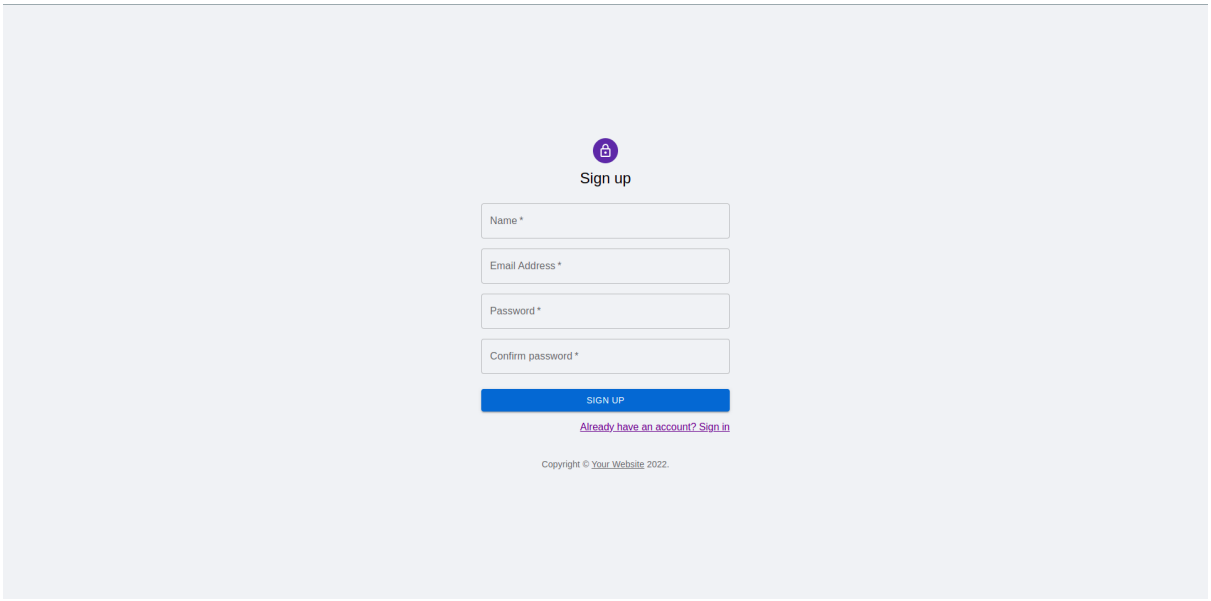


Registrácia nového používateľa

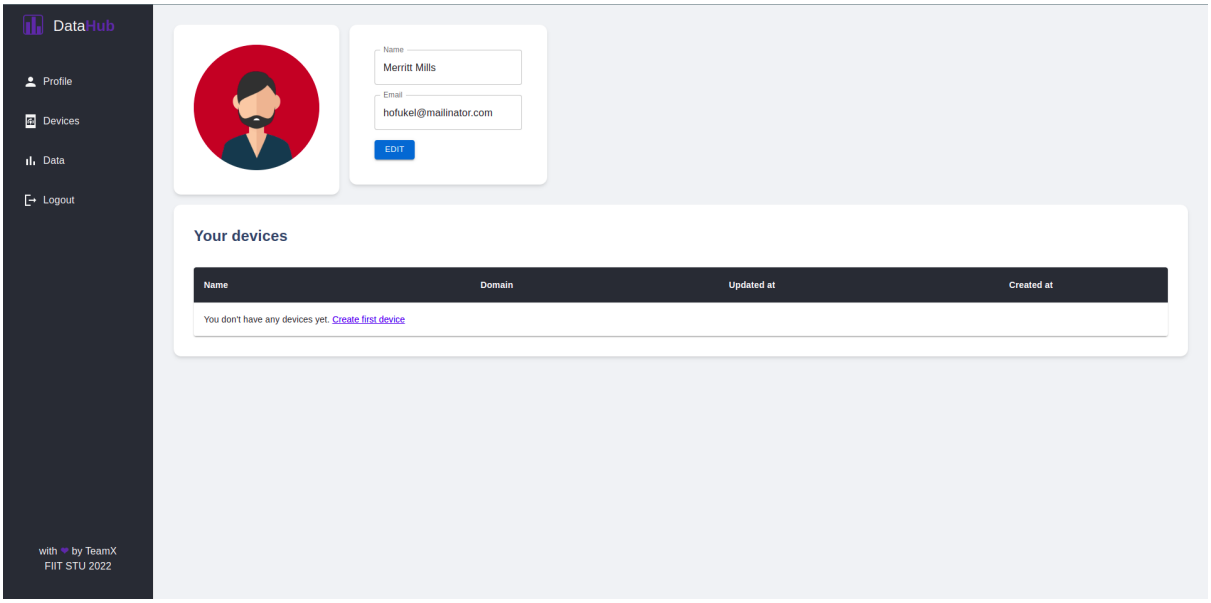
Na registráciu nového používateľa je potrebné navštíviť prihlasovaciu obrazovku a stlačiť odkaz pod formulárom “***Don't have account? Sign up***”.



Prehliadač otvorí registračný formulár, všetky polia tohto formulára sú povinné. Po vyplnení je potrebné stlačiť tlačítko - “**SIGN UP**”.



V prípade úspešnej registrácie systém automaticky prihlasuje používateľa a zobrazí hlavnú stránku aplikácie.

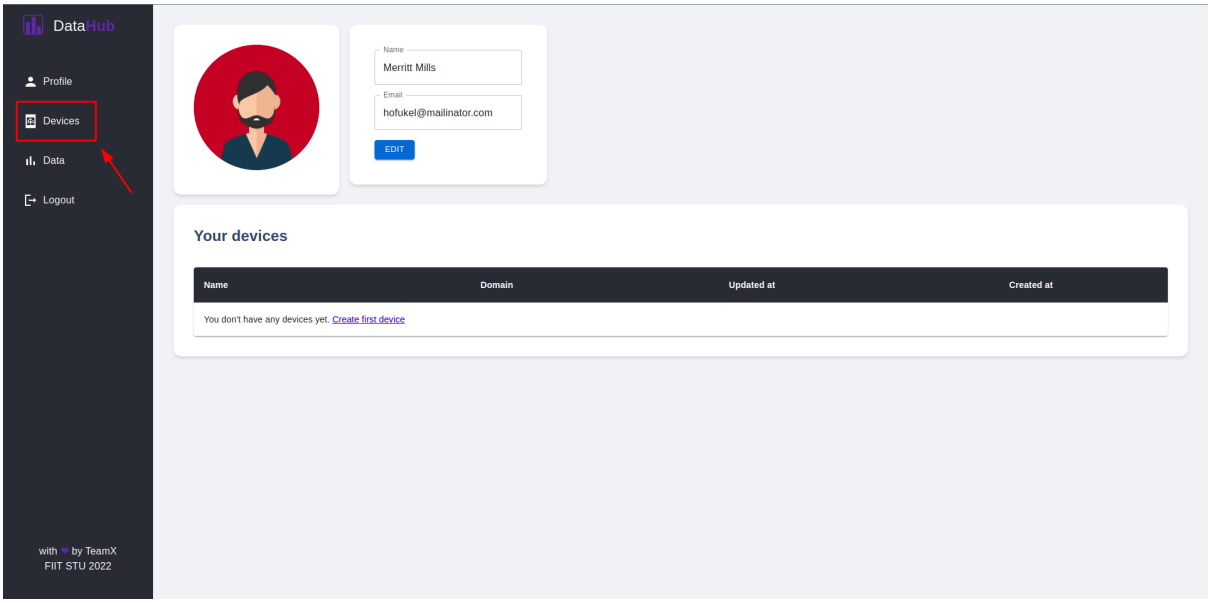


Prihlásenie používateľa

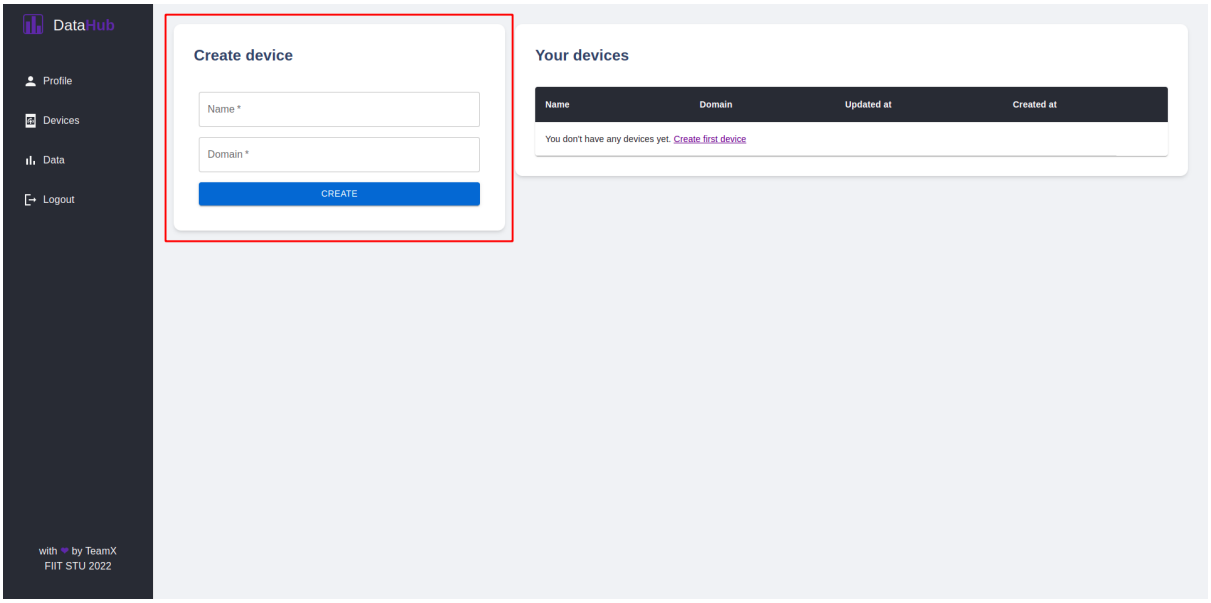
Na prihlásenie je potrebne navštíviť hlavnú stránku systému, vyplniť prihlasovací formulár a stlačiť tlačítko - “**SIGN IN**”. Po úspešnom prihlásení systém presmeruje užívateľa na hlavnú stránku aplikácie.

Vytvorenie nového zariadenia

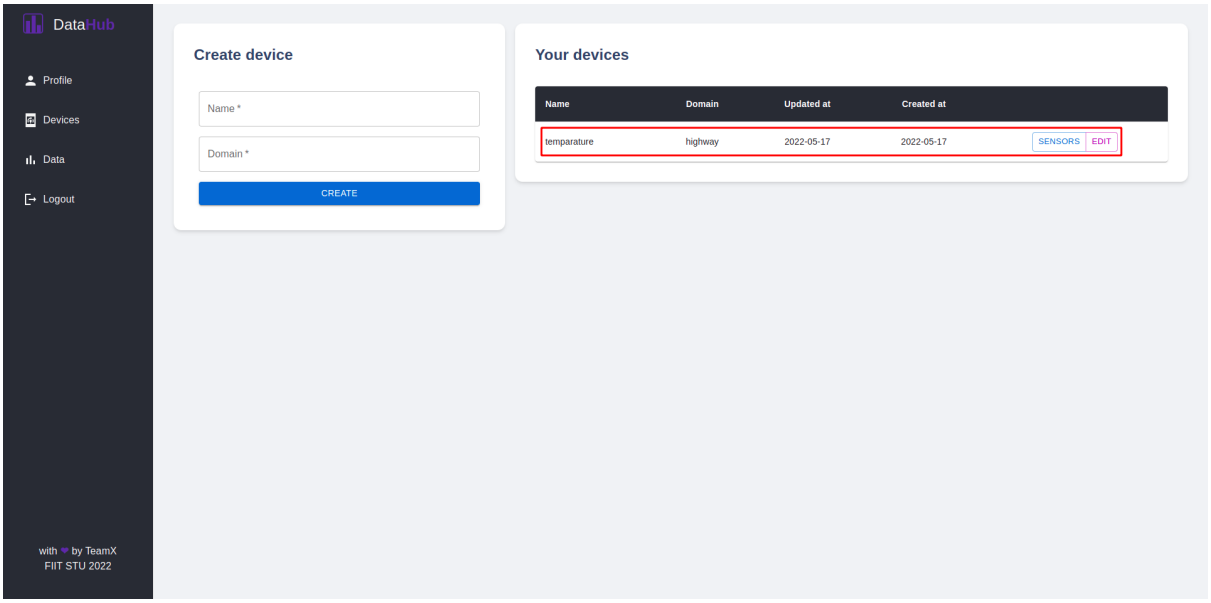
Na vytvorenie nového zariadenia je potrebné sa prihlásiť, po prihlásení v menu na hlavnej obrazovke treba vybrať sekciu “**Devices**”



Na vytvorenie nového zariadenia v sekcii “**Devices**” treba vyplniť formulár vytvorenia zariadenia. Všetky polia sú povinné.

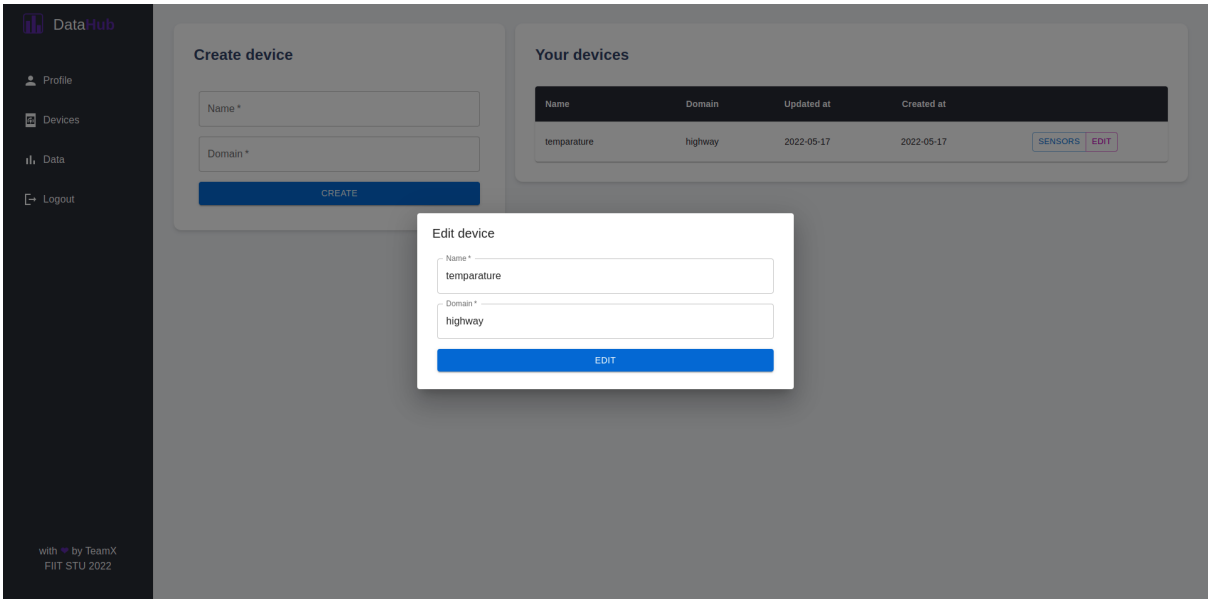


Po úspešnom vytvorení sa objaví nové zariadenie v tabuľke “**Your devices**”



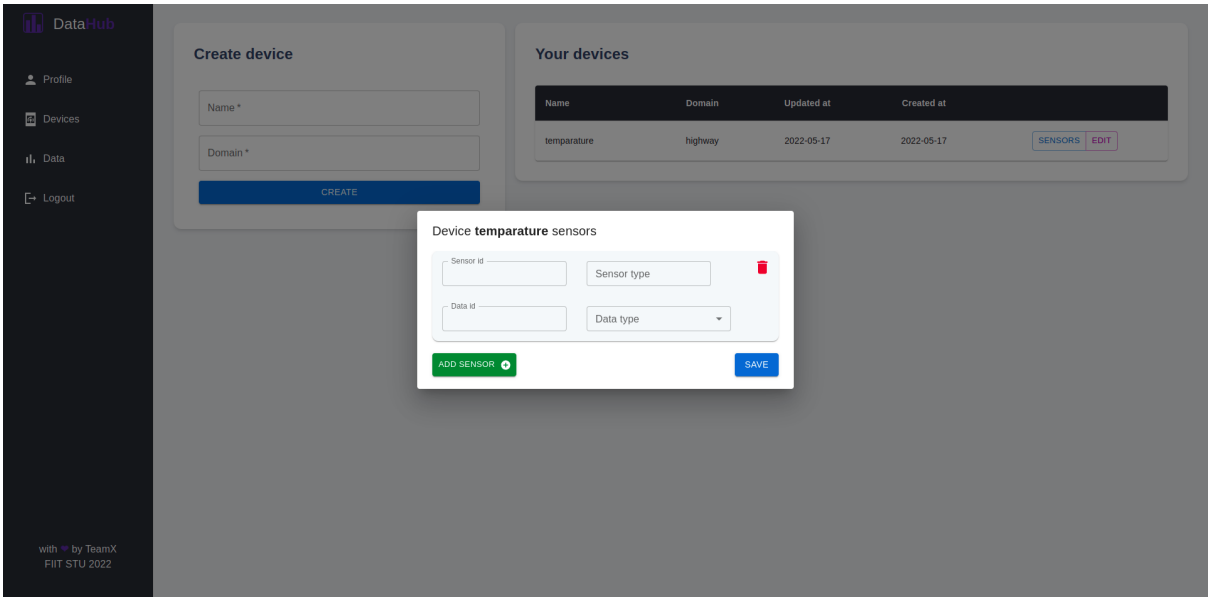
Editácia zariadenia

Pre editáciu zariadenia treba prejsť na “**Device**” sekciu a stlačiť tlačítko “**EDIT**” vedľa príslušného zariadenia. Po stlačení sa objaví formulár na editáciu zariadenia.



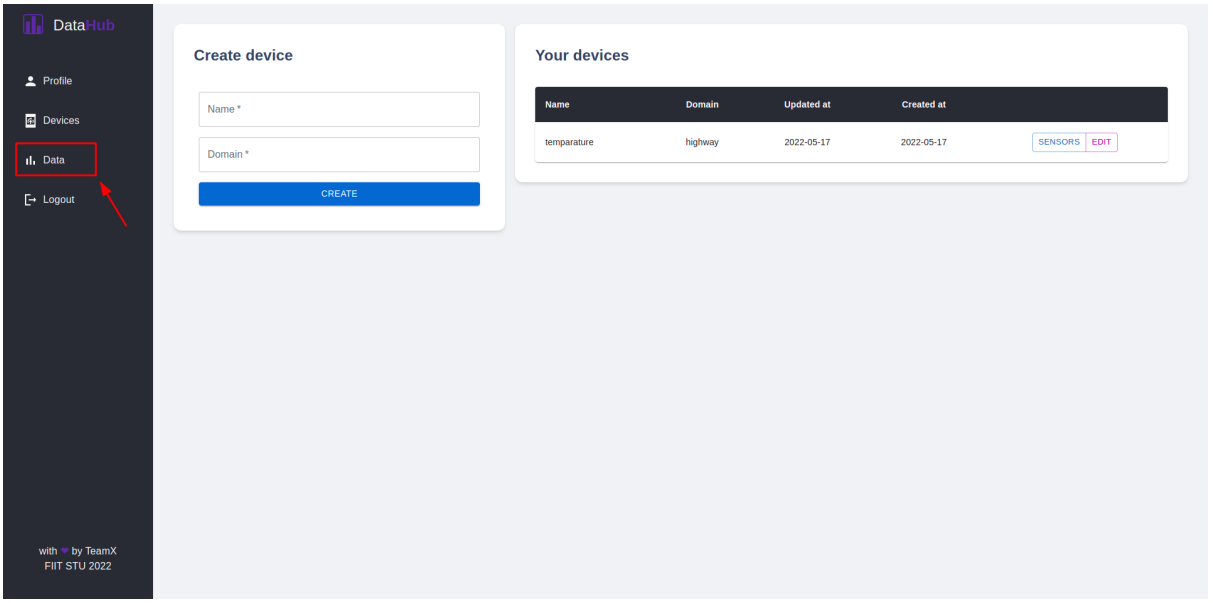
Pridanie / odstránenie senzorov zariadenia

Pre pridanie / odstránenie senzorov zariadenia treba prejsť na “**Device**” sekciu a stlačiť na tlačítko “**SENSORS**” vedľa príslušného zariadenia. Po stlačení sa objaví formulár na editáciu senzorov.

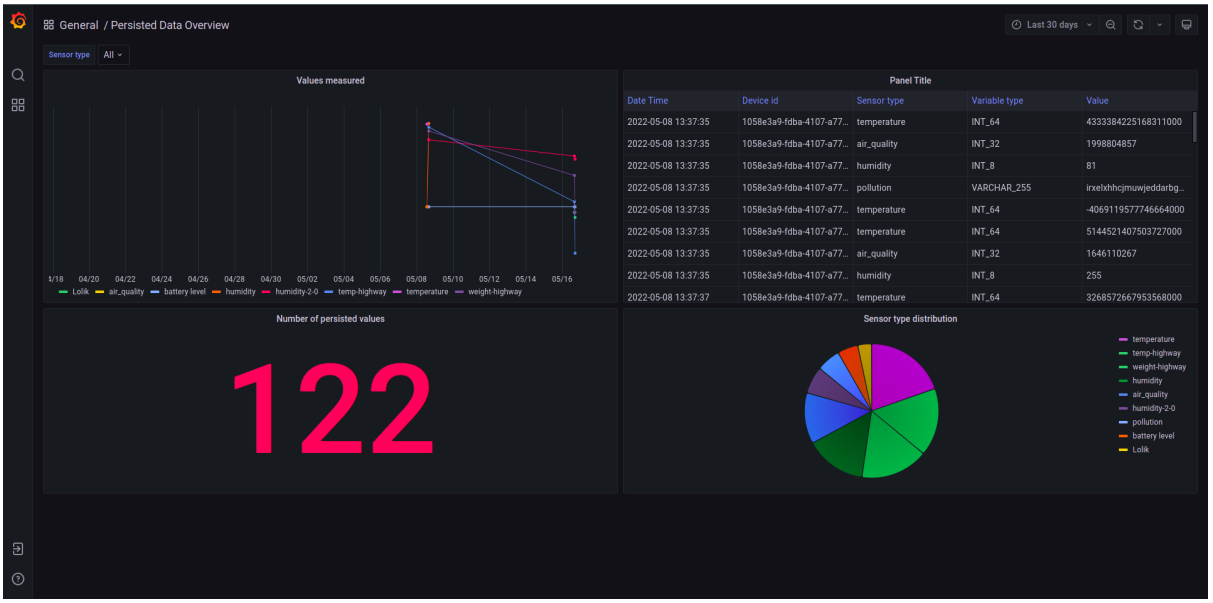


Zobrazenie prehľadu dát

Na zobrazenie prephľadu dát treba prejsť na sekciu “**Data**”.



Po stlačení sa otvorí “**Grafana**”, dokumentácia: (https://grafana.com/docs/grafana/latest/)


General / Persisted Data Overview
Last 30 days
Sensor type
All
Values measured
4/18 04/20 04/22 04/24 04/26 04/28 04/30 05/02 05/04 05/06 05/08 05/10 05/12 05/14 05/16
Lolik air_quality battery level humidity humidity-2-0 temp-highway temperature weight-highway
Panel Title
Date Time Device id Sensor type Variable type Value
2022-05-08 13:37:35 1058e3a9-fdba-4107-a77... temperature INT_64 433338422516831100
2022-05-08 13:37:35 1058e3a9-fdba-4107-a77... air_quality INT_32 1998804857
2022-05-08 13:37:35 1058e3a9-fdba-4107-a77... humidity INT_8 81
2022-05-08 13:37:35 1058e3a9-fdba-4107-a77... pollution VARCHAR_255 irxelxhhcjmuwjeddarbg...
2022-05-08 13:37:35 1058e3a9-fdba-4107-a77... temperature INT_64 -4069119577746664000
2022-05-08 13:37:35 1058e3a9-fdba-4107-a77... temperature INT_64 5144521407503727000
2022-05-08 13:37:35 1058e3a9-fdba-4107-a77... air_quality INT_32 1646110267
2022-05-08 13:37:35 1058e3a9-fdba-4107-a77... humidity INT_8 255
2022-05-08 13:37:37 1058e3a9-fdba-4107-a77... temperature INT_64 326857266795356800
Number of persisted values
122
Sensor type distribution
temperature
temp-highway
weight-highway
humidity
air_quality
humidity-2-0
pollution
battery level
Lolik