TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Madis Põld 213123IAIB

Interaktiivne veebivormi generaator kahe API vastavusseoste koostamiseks ja talletamiseks

Bakalaureusetöö

Juhendaja: Vahur Kotkas
MSc

Tallinn 2022

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Madis Põld

14.03.2022

Annotatsioon

Lõputöö eesmärgiks on arendada rakendus, mis võimaldab luua kahe etteantud API spetsifikatsiooni põhjal vastavusseoste komplekti, et oleks võimalik andmeid ühest andmekogust teise transportida ja seda dokumenteerida.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 51 leheküljel, 7 peatükki, 26 joonist, 1 tabel.

Abstract

Interactive Web Application to Generate Mapping Definitions for Data Transformations Between APIs

The purpose of this thesis is to develop an interactive web application that enables the creation of data transformation mappings for APIs based on given API specifications and user input.

The thesis is in Estonian and contains 51 pages of text, 7 chapters, 26 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application programming interface,* rakendusliides, mis võimaldab rakendusel suhelda teiste rakendustega
REST	*Representational State Transfer,* tarkvaraarhitektuuri laad, mis seab veebirakenduse loomisele kindlad piirid
JSON	*Javascript Object Notation,* standardne inimloetava teksti kujul andmeformaat
SaaS	*Software as a service,* tarkvara kui teenus (tellimusmudel)
XML	*Extensible markup language,* standardne inimloetava teksti kujul andmeformaat.
MVP	*Minimal viable product,* minimaalne elujõuline toode
Mock	*Mock (service),* näidis (teenus) loodud vastavalt reaalsele viimase testimiseks
ADF	*Azure Data Factory,* microsofti pilveteenus
DRY	*Dont repeat yourself,* arendusprintsiip
KISS	*Keep it simple, stupid,* disainiprintsiip
PostgreSQL	*Database management system,* relatsioonilise andmebaasi haldussüsteem
OS	*Open source,* avatud lähtekoodiga
JRE	*Javascript runtime environment,* keskkond javascripti jooksutamiseks
VPS	*Virtual private server,* virtuaalne privaatserver
CI/CD	*Continuous integration, continuous deployment* – automaatne uuenduste integreerimine
CLI	*Command line interface,* käsurida
Bug	*Code bug,* viga koodis
Boilerplate	*Standardized basis,* standardiseeritud taaskasutatav põhi tekstis/koodis
Backend	*Backend layer,* andmeedastuskiht
Frontend	*Frontend layer,* esitluskiht
SQL	*Structured Query Language,* struktureeritud päringukeel andmebaasioperatsioonide teostamiseks

UI	*User interface,* kasutajaliides
MUI	*Material UI,* komponentide teek
WCAG	*Web content accessibility guidelines,* juurdepääsetavuse juhised veebile
SSD	*Solid-state drive,* kõvaketas, püsimälu
RAM	*Random access memory,* muutmälu
Nginx	*Web server,* veebiserver
DMS	*Database management system,* andmebassi haldussüsteem
YAML	*YAML,* andmete serialiseerimiskeel konfiguratsioonifailide tarbeks
DTO	*Data transfer object,* päringu kehandi struktuur
CORS	*Cross origin resource sharing,* poliis, mis defineerib brauseris lubatud ressursside laadimise piirangud domeenipõhiselt
ISO	*International Organization for Standardization,* Rahvusvaheline Standardiseerimise Organisatsioon
Modbus	*Internet protocol,* Andmevahetusprotokoll elektroonikaseadmetele.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva lõputöö raames luuakse rakendus teenuste vaheliste seoste loomiseks ja dokumenteerimiseks, et võimaldada andmevahetust erinevate allikate vahel. Olgu nendeks allikateks näiteks ilmaandmed, börsihinnad, teeolud või mõõteriistad. Üks probleem andmete liigutamisel erinevate andmekogude vahel on seoste läbipaistvus. Näiteks kui on vaja salvestada mitme päringuga kogutud andmed mitmesse tabelisse teises teenuses, mis nõuab järjestikku sooritatud üksteisest sõltuvaid päringuid. Selle protsessi seadistamine ja haldamine on üsna keerukas. Seda aitab lihtsustada arendatav rakendus vastavusseoseid illustreeriva kasutajaliidese ja genereeritud andmevahetuse definitsioonidega.

Probleemi püstitus tekkis vajadusest parandada protsessi andmete kogumiseks erinevatest allikatest kasutades seejuures erinevaid suhtlusprotokolle nagu näiteks REST ja Modbus TCP. Eesmärk on luua maksimaalselt abstraktne rakendus, et tagada laiapõhine kasutusmugavus. Käesoleva lõputöö raames ei realiseerita päringute sooritamist, vaid genereeritakse Autori poolt välja töötatud formaadis vastavusseoste definitsioonid JSON kujul. Töös kirjeldatakse loodud lahendust ilmaandmete näitel.

Lõputöö sisuks on rakenduse arendamine, mis võimaldab luua kahe etteantud API spetsifikatsiooni põhjal vastavusseoste komplekti, et oleks võimalik andmeid ühest andmekogust teise transportida ja seda dokumenteerida. Töö raames tuleb valida või luua sobilik APIde spetsifitseerimiskeel, disainida vastavusseoste esitus JSON kujul, realiseerida generaator, mis API-de spetsifikatsioonide alusel koostab sobiva veebipõhise kasutajaliidese, ja päringud andmete hankimiseks ja talletamiseks. Veebivormil andmete sisestamine ja seoste loomine peab olema võimalikult kasutajasõbralik.

2 Nõuete analüüs

Antud peatükis kirjeldatakse rakendusele kohalduvad nõuded.

2.1 Üldised nõuded

Rakendus peab olema võimeline eri formaatides kirjeldatud API teenuseid mõistma ja sisendina töötlema. See tagab laiema kasutatavuse ja ühilduvuse. [1]

Rakenduses tehtud tehingud peavad olema teostatud standardiseeritud spetsifikatsioonidele vastavate API kirjelduste põhjal. See aitab minimeerida sisestusel tekkivate ja rakenduse töös esinevate vigade arvu. Suur probleem transformeerimisel on andmete korrektsus ja sisestusel tehtud vead, mille vältimiseks on kindlasti vaja andmete valideerimist. [1]

Rakendus peab olema realiseeritud REST arhitektuuris, mis tagab laialdase ühilduvuse ja paindliku arhitektuuri.

2.2 Funktsionaalsed nõuded

Süsteemi funktsionaalsed nõuded:

- Süsteemis saab laadida üles kindlas spetsifikatsioonis API teenuse
- Süsteemis saab vaadata üles laetud API teenuse otspunkte.
- Süsteemis saab salvestada API teenuse otspunkte
- Süsteemis saab kustutada salvestatud API teenuse otspunkte
- Süsteemis saab salvestatud API teenustega vastavusseoseid luua
- Süsteemis saab genereerida vastavusseoste definitsioone

- Süsteemis saab salvestada vastavusseoste definitsioone ja nendega uusi vastavusseoseid luua.

2.3 Mittefunktsionaalsed nõuded

Süsteemi mittefunktsionaalsed nõuded:

- Veebirakenduse kasutajaliides peab vastama vähemalt WCAG 2.1 tasemele AA
- Rakendus peab olema võimeline kasutama keskkonnamuutujaid
- Keskkonnamuutujate nimed peavad olema sisulised
- Rakendust peab saama hoiustada ilma ümber programmeerimata eri domeenidel ja serverites (dünaamiliste keskkonnamuutujatega tagatud)
- Rakendus peab olema realiseeritud komponendi-põhiselt

3 Sarnased rakendused ja teenused

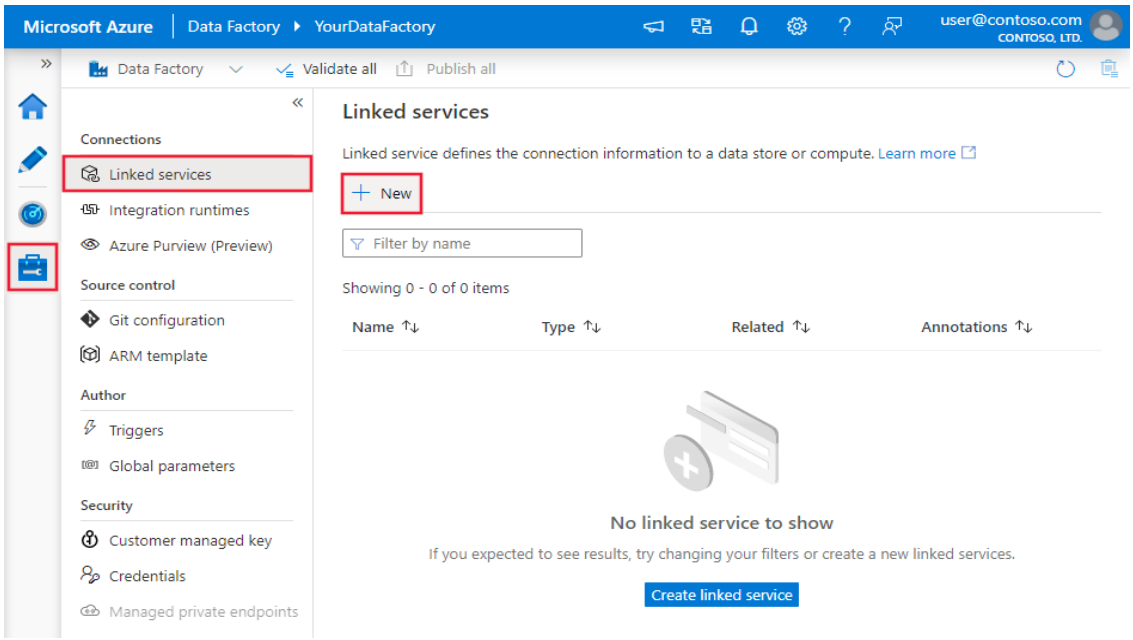
Antud peatükis kirjeldatakse uuringu tulemusena avastatud arendatavale rakendusele sarnased tooted ja teenused. Sarnast probleemi on varasemalt proovitud eri rakendustes juba lahendada ja seda ka edukalt. Uuringu tulemusel jäid silma just Azure Data Factory ja Amazon Web Services pakutud lahendused.

3.1 Azure Data Factory

Microsofti pilveteenus Azure Data Factory (ADF) pakub andmete integratsiooni ja transformeerimise lahendusi mitmel kujul, nende hulgas ka REST API otspunktide vahelisi andmete liigutamisi. [2]

ADF võimaldab kopeerida andmeid kasutades GET ja POST meetodeid ja neid salvestada POST, PUT, PATCH meetoditega. Samad kopeerimise piirangud sai võetud ADF näitel kasutusse ka enda rakenduses. Autoriseerimisel on tugi loodud Basic, AAD ja süsteemi sisemise kasutaja jaoks. Paginatsioon on võimaldatud. Päringu vastuse formaadina on lubatud vaid JSON. Seoste loomiseks ja päringute sooritamiseks kasutab Azure kopeerimise tegevusi (*Copy activity*), mis on vaja seadistada allikate *(sources)* salvestamiseks andmehoidla (*sink data store*) ja transformeerimise kirjeldusega. [2]

ADF pakub sarnast funktsionaalsust nagu lõputöö raames kavandatud rakendus koos päringute sooritamise ja autoriseerimisega, kuid ei võimalda samaväärset seoste visualiseerimist. Lisaks on ADF tasuline teenus ja nõuab kuumaksega Azure tellimust. Samuti on ADF andmete transformeerimise teenuste kasutamiseks vaja nende pilveplatvormil andmehoidla seadistada ja integreerimine väliste teenustega on keerukam. [3]



Joonis 1. Azure Data Factory kasutajaliides.

3.2 Amazon Web Services

Amazon Web Services (AWS) pilveplatvorm pakub samuti andmete transformeerimise teenust REST APIdele. Andmete muundamiseks kasutatakse kaardistamise malle (*mapping template*), mille abil on võimalik päringu vastus muuta vastavalt mallile sobivale kujule. [4]

Kaardistamise mallid pakuvad suurt paindlikkust ja võimaldavad kasutada ka tsükleid (nt. *foreach)*, et massiivi kujul päringu kehand ümber struktureerida. Sellele järgnevalt on võimalik seadistada integratsiooni päring, mille abil uuel kujul andmeid tarbida. Päringusse on võimalik ka manuaalselt muid muutujaid lisada AWS keskkonna kontekstist. AWS andmete transformeerimine on paindlikum kui meie arendatav rakendus, kuid ei anna võrdväärset visuaalset ülevaadet loodud seostest. Koodi kujul seoste loomine nõuab väga laialdasi teadmisi teenuste kohta transformatsiooni kirjeldamise hetkel. ADFile sarnaselt on ka AWS tasuline teenus. [4] [5]

Input mapping template (photos example)

The following is the input mapping template that corresponds to the original JSON data for the photos example:

```
#set($inputRoot = $input.path('$'))
{
  "photos": {
    "page": $inputRoot.photos.page,
    "pages": "$inputRoot.photos.pages",
    "perpage": $inputRoot.photos.perpage,
    "total": "$inputRoot.photos.total",
    "photo": [
#foreach($elem in $inputRoot.photos.photo)
      {
        "id": "$elem.id",
        "owner": "$elem.owner",
        "secret": "$elem.secret",
        "server": "$elem.server",
        "farm": $elem.farm,
        "title": "$elem.title",
        "ispublic": $elem.ispublic,
        "isfriend": $elem.isfriend,
        "isfamily": $elem.isfamily
      }#if($foreach.hasNext),#end

#end
    ]
  }
}
```

Joonis 2. AWS kaardistamise mall.

4 Kasutatud tehnoloogiad ja metoodikad

Antud peatükk kirjeldab rakenduse arendamisel kasutatud tehnoloogiad ja metoodikad ja selgitab miks just nii sai otsustatud.

4.1 Metoodikad ja printsiibid

Arenduse käigus on võetud eesmärgiks järgida fundamentaalseid printsiipe, mis aitavad tagada parema koodi kvaliteedi ja jätkusuutlikkuse.

Üheks neist on DRY printsiip, mille eesmärk on vähendada korduseid tarkvara mustrites. Uues kohas varasemalt kirjutatud koodijuppi rakendades, laienda olemasolevat nõnda, et sa saad seda ilma uut koodi kirjutamata või varasemat dubleerimata kasutada. [6]

Teine printsiip, millest kinni pidada on KISS. KISS printsiip ütleb, et enamus süsteemid töötavad töökindlamalt ja paremini kui need on disainitud võimalikult lihtsalt. Sellest tulenevalt on eesmärk kujundada rakendus võimalikult primitiivselt. [7]

4.2 Tehnoloogiad ja arhitektuur

Tervikliku rakenduse moodustavad PostgreSQL andmebaas, NodeJS backend teenus, mis ehitatud NestJS raamistikuga ja ReactJS frontend rakendus.

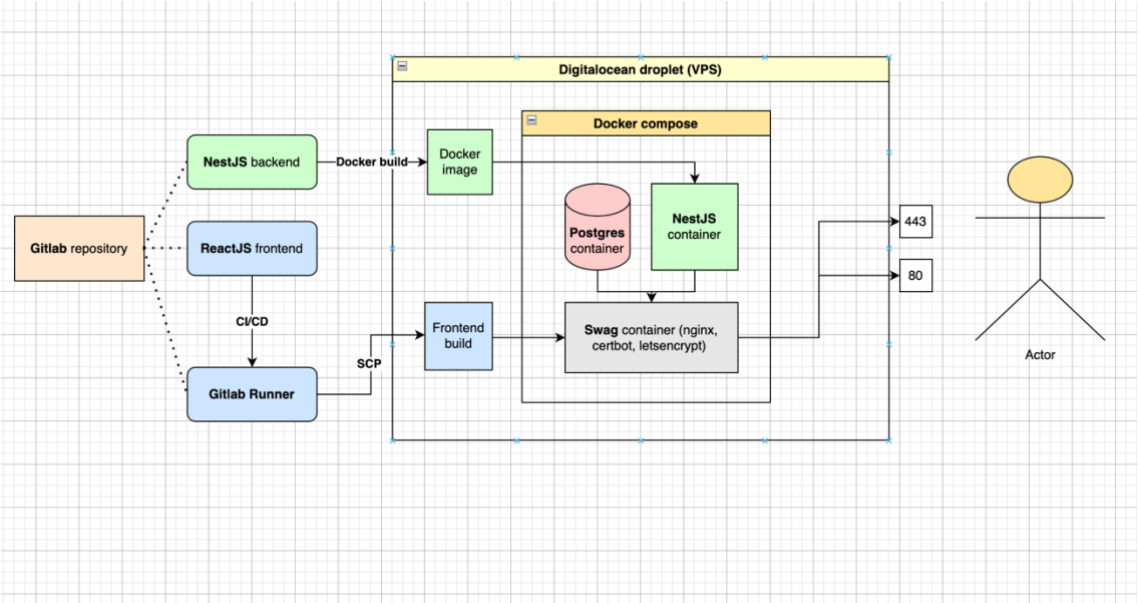
Rakenduse REST API kirjelduskeeleks on OpenAPI (Swagger), mis on REST teenuste kirjeldamises kujunenud IT valdkonna standardiks. OpenAPI üheks tugevuseks on kindlasti stabiilsus ja disainisuunitlus. Nimelt on võimalik enne teenuse reaalselt implementeerimist testida enda soovitud lahendust näidete ning *mock* teenuste peal. Lisaks sellele on OpenAPI eeliseks laialdane aktiivsete arendajate hulk, mille tõttu on tööriistu ka rohkem. Tööriistadest tulenev paindlikkus on ka meie rakenduse osas väga kriitiline. [8] [9]

Paigaldatud rakendus on hoiustatud Ubuntu 20.04 LTS x64 baasil loodud VPS *(*virtuaalses privaatserveris) kasutades jooksutamiseks Dockeri konteinerlahendust. Domeen on tellitud ja hallatav veebimajutus.ee platvormil. Automaatne keskkonna uuendamine (CI/CD) on realiseeritud Gitlab CI/CD konfiguratsiooni abil kasutades jagatud TTÜ Gitlab jooksutajaid *(runner)*.

Paigaldatud rakenduses on veebiserver seadistatud *reverse-proxy* vastupidise vahendusserverina, mis serveerib peamiselt domeenilt (apitransformations.ee) ReactJS frontend veebirakenduse ja „api“ eesliitega alamdomeenilt NestJS backend rakenduse.

4.2.1 Komponentide diagramm

Järgnev komponentide diagramm illustreerib töötavat rakendust ja selle protsesse. (Joonis 3)



Joonis 3. Komponentide diagramm.

4.3 PostgreSQL andmebaas

Rakenduse kaardistamisel sai otsustatud PostgreSQL andmebaas kasutusele võtta eelkõige varasema kogemuse ja laia kasutajaskonna tõttu. Lisaks on PostgreSQL andmebaasis SQL standardite vastavus üks kõrgemaid OS andmebaasihaldussüsteemidest. Samas tuleb tõdeda, et samaväärse lahenduse saab realiseerida ka teiste andmebaasisüsteemidega. Rakenduse andmebaas on vajalik NestJS backend teenuse toimimiseks ja kirjete talletamiseks. [10]

4.4 NestJS backend teenus

Rakenduse kohaliku REST teenusena on realiseeritud NestJS raamistikuga ehitatud NodeJS teenus. Tehnoloogiate valikul leidsin, et on oluline hoida koodibaas ühes keeles,

et tagada suurem läbipaistvus ja ühtsus mõlemal poolel (*backend* ja *frontend*). Selleks osutus Javascript ja otsesemalt viimase edasiarendus Typescript.

Typescript on hetkel üks kiiremini populaarsust saavutav tehnoloogia veebiarenduses ja sedasi on olnud juba aastaid. Uurijad on leidnud, et Typescripti tuvastab 15% levinud *bugidest* varajaselt kompileerimise faasis, mis säästab nii arendaja kui testija aega. [11]

NestJS osutus valituks tema väikese õppekurvi ja mugava ning põhjaliku CLI tõttu. Nimelt saab NestJS käsurea tööriista abil hõlpsasti genereerida kontrollereid, mudeleid, teenuseid ja palju muud, rääkimata rakenduse *boilerplate* genereerimisest. [12] [13]

Teenus avab kasutajaliidesele kasutamiseks Endpoint, Service ja Proxy kontrollerid. API otspunktide halduseks on Endpoint kontrolleril realiseeritud GET, POST ja DELETE meetodid. Samad meetodid on realiseeritud ka Service kontrolleril API teenuste halduseks. Proxy kontroller on kasutusel väliste teenuste suunal päringute sooritamiseks. Ilma selleta seab brauser rakendusele CORS piiranguid ja takistab päringute õnnestumist.

NestJS ühendub PostgreSQL andmebaasiga TypeORM kaudu. ORM on tänapäeval üks põhilisi tehnikaid kuidas relatsioonandmebaaside struktuuri seostada objektorienteeritud keele objektidega. Sisuliselt on tegu abstraktsiooniga Andmebaasi ja Backend teenuse kohal, et ei peaks manuaalselt SQL päringuid kirjutama. NestJS soovitab ja on liidestunud just TypeORMiga kuna see on kõige küpsem ORM Typescripti jaoks. [14]

Järgnevalt on toodud välja rakenduses kasutatud tabelite definitsioonid. (Joonis 4, Joonis 5)

```
iaib=# \d+ "Endpoint"
                                                  Table "public.Endpoint"
   Column    |       Type        | Collation | Nullable |                Default                | Storage  | Stats target | Description
-------------+-------------------+-----------+----------+---------------------------------------+----------+--------------+-------------
 id          | integer           |           | not null | nextval('"Endpoint_id_seq"'::regclass) | plain    |              |
 rootUrl     | character varying |           | not null |                                       | extended |              |
 method      | character varying |           | not null |                                       | extended |              |
 path        | character varying |           | not null |                                       | extended |              |
 parameters  | json              |           |          |                                       | extended |              |
 response    | json              |           |          |                                       | extended |              |
 requestBody | json              |           |          |                                       | extended |              |
Indexes:
    "PK_592a6f6e88ba9d6e4ed114b2e90" PRIMARY KEY, btree (id)
```

Joonis 4. Endpoint tabeli definitsioon.

```
iaib-# \d+ "Service"
                                                  Table "public.Service"
    Column     |       Type        | Collation | Nullable |                 Default                 | Storage  | Stats target | Description
---------------+-------------------+-----------+----------+-----------------------------------------+----------+--------------+-------------
 id            | integer           |           | not null | nextval('"Service_id_seq"'::regclass) | plain    |              |
 rootUrl       | character varying |           | not null |                                         | extended |              |
 name          | character varying |           | not null |                                         | extended |              |
 specification | character varying |           | not null |                                         | extended |              |
 definition    | json              |           |          |                                         | extended |              |
Indexes:
    "PK_c77cf540affcc8a04962fc6e9f8" PRIMARY KEY, btree (id)
```

Joonis 5. Service tabeli definitsioon.

4.5 ReactJS Frontend rakendus

Frontend tehnoloogiana sai võetud kasutusele ReactJS tema uuenduslikkuse, laia kommuuni ja arvamusvabaduse *(unopinionated)* printsiibi tõttu. React teeb väga vähesed valikuid arendajate eest ära ja annab suure vabaduse ise otsustada raamistiku sisemiste tehnoloogiate ja arhitektuuri osas. Samas lisab see ka vastutust teha õigeid otsuseid. [15]

Teisest küljest Google välja töötatud Angular raamistik seab päris kindlad reeglid tehnoloogiate, arhitektuuri ja koodi tükeldatuse osas. Kood jaguneb Angulari mooduliteks ja Typescript kasutamine on kohustuslik koos mitmete muude tehnoloogiatega.

Käesoleva töö arendust planeerides oli palju teadmatust kasutusjuhtude ja protsesside analüüsimisel mille tõttu sai valitud ReactJS tema paindlikkuse ja väiksemate rakendusele kohalduvate nõuete tõttu. Lisaks on Reactil eri frontend raamistikest üks suurema arendajate toega kommuun, mis tagab rohke kolmandate osapoolte arendatud teekide hulga ja suurema tõenäosuse probleemidele kiire lahenduse leidmiseks. [16]

Jagatud komponentide teegina on kasutajaliideses kasutusel Material UI (MUI). MUI sai valitud suurepärase WCAG toe, seejuures ka väga intuitiivse olekuseisundite kuvamise, tõttu. Lisaks on MUI äärmiselt põhjaliku dokumentatsiooniga. Komponentide kasutusele võtmine on lihtne ja variatsioone lõpmata palju. [17]

ReactJS rakenduses on kasutusel funktsionaalsed komponendid ilma varasemalt standardse klassipõhise struktuurita. Sel viisil on minu arvamusel kood konkreetsem. Lisaks on järgitud komponentide põhist arhitektuurimustrit, mille alusel on kood jaotatud rumalateks ja tarkadeks komponentideks eesmärgipõhiselt. Rumalate komponentide eesmärk on vaid kuvamine. Nad ei muuda olekut (kui mitte enda oma) ega tee muid

toiminguid. Nende hulka kuuluvad ka *layout*-komponendid nagu Header ja Footer. Targad komponendid (st. konteinerkomponendid) koosnevad rumalatest komponentidest ja tegelevad olekumuutuste ja päringute sooritamise ja palju muuga.

Rakenduse *runtime* oleku hoidmiseks on kasutusel React Context, mis võimaldab üle kogu Reacti DOMi puu, kus Context on defineeritud, jagada olekut seal olevate komponentide vahel. Rakenduses hoiustatakse seal API teenuseid, otspunkte ja vastavusseoste definitsioone. [15]

4.6 Digitalocean Ubuntu virtuaalserver

Digitalocean on serveripinna rentija 14 erineva globaalse andmehoidlaga, kes pakub muuhulgas riistvaraliselt ja võimekuselt konfigureeritavaid virtuaalseid privaatservereid (VPS). Digitaloceani kuluefektiivsus ja läbipaistvus tellimuse tegemisel osutus kaalupunktiks nende valimisel. Paigaldatud rakendus jookseb Ubuntu serveril 1 GB muutmäluga, 25 GB SSD ja jagatud CPUga, mis on kuutasuga 5$ (Joonis 6). Digitaloceani kasuks räägib ka piirangute puudumine serveri kasutajale ja täielik seadistusvabadus.

Joonis 6. Digitalocean VPS statistika.

4.7 Gitlab CI/CD

Gitlab CI/CD tööriist võimaldab hõlpsasti seadistada automaatset integratsiooni ja paigaldust koodibaasis tehtud uuenduste korral. Boonusena võimaldab CI/CD püüda ka kiirelt arendusel tehtud vigu, mis kompileerides või paigaldades avalduvad. Automaatne integratsioon on seadistatud läbi Gitlab CI/CD tööriista, sest Taltechi koodivaramu on samuti Gitlabis. [18]

Frontend rakenduse CI/CD konfiguratsioonifail on kahe etapiga (Joonis 7):

- Build
    1. Ehitab ja pakib kokku frontend toodangu koodi
    2. Salvestab koodi artefaktina Gitlabi keskkonda tunniks ajaks
- Deploy
    1. Ühendub keskkonnamuutujates defineeritud serverisse üle SSH (siinkohal meie DigitalOcean VPS)
    2. Peatab rakenduse ja liigutab SCP abil eelmises etapis salvestatud artefakti serverisse.
    3. Taaskäivitab rakenduse

```yaml
variables:
  SSH_PRIVATE_KEY: $SSH_PRIVATE_KEY
  SSH_USER: $SSH_USER
  REMOTE_SERVER: $REMOTE_SERVER

stages:
  - build
  - deploy

cache:
  paths:
    - node_modules/

build:
  image: node:16.14-alpine
  stage: build
  only :
    - main
  script:
    - cd frontend
    - npm install
    - npm run build
  artifacts:

    paths:
      - frontend/build
    expire_in: 1 hour

deploy:
  image: ubuntu
  stage: deploy
  only:
    - main
  before_script:
    - 'command -v ssh-agent >/dev/null || ( apt-get update -y && apt-get install openssh-client -y )'
    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
  script:
    - ssh -o StrictHostKeyChecking=no $SSH_USER@$REMOTE_SERVER "docker-compose down"
    - ssh -o StrictHostKeyChecking=no $SSH_USER@$REMOTE_SERVER "rm -rf /root/data/frontend/*"
    - scp -o StrictHostKeyChecking=no -r frontend/build/** $SSH_USER@$REMOTE_SERVER:/root/data/frontend
    - ssh -o StrictHostKeyChecking=no $SSH_USER@$REMOTE_SERVER "docker-compose up -d"
```

Joonis 7. Gitlab CI/CD konfiguratsioonifail gitlab-ci.yml.

4.8 Docker ja paigaldus

Rakendus on seadistatud serveris jooksma Docker konteineritena Docker Compose konfiguratsiooni põhjal. Docker sai valitud stabiilsuse ja keskkonna agnostilisuse tõttu. Nimelt vähendab Dockeril loodud lahendus konfigureerimist liikudes keskkonnast keskkonda ja CI/CD uuenduste järgselt on lihtne rakenduse komponente taaskäivitada. Docker konteiner on *image* (malli) baasil loodud jooksutatav iseseisev tükk tarkvara, mis ei sõltu keskkonnast kus ta jookseb. [19]

Docker Compose on tööriist orkestreerimaks ja jooksutamaks mitme konteineri koostöös töötavaid rakendusi. Seadistuse konfiguratsioon on YAML faili kujul, mis defineerib konteineritele ühise võrgu, volüümid ja individuaalsed atribuudid (sh. mall, avatud

pordid, volüümid). Konteineri volüümid on tema loodud või tema poolt kasutatud andmete säilitamiseks määratud seosed serveri failisüsteemi ja konteineri vahel. [20]

Rakenduse Docker Compose seadistusfail (Joonis 8) sätestab 3 konteinerit:

- Swag
    - Nginx veebiserver, mis on seadistatud *reverse-proxy* vaheserverina serveerima frontend rakendust juurdomeenilt apitransformations.ee ja backendi alamdomeenilt eesliitega api.apitransformations.ee
    - SSL sertifikaatide haldaja ja automaatne uuendaja. Kontrollib igal õhtul kas sertifikaatide aegumiseni on jäänud vähem kui 30 päeva ja kui on, siis uuendab neid.
    - Omab kahte volüümi: frontend rakendus ja swag konfiguratsioon.
    - Avatud pordid 80 ja 443 välismaailmale
- Postgres
    - PostgreSQL andmebaasi konteiner
    - Säilitab andmebaasi seisu volüümis, et säilitada andmeid katkestuse või taaskäivituse järgselt
    - Avatud port 5432
- Nestjs
    - NestJS backend rakenduse konteiner ehitatud madispold/apitransformations:nestjs *image* (malli) baasil, mis omakorda loodud backend rakenduse kaustas oleva Dockerfile põhjal.
    - Avatud port 8000

Konteinerite keskkonnamuutujad on sätestatud konfiguratsioonifailides vastavalt swag.env, postgres.env ja nestjs.env. NestJS keskkonnamuutujate failis on viide andmebaasile HOST muutujas vaid „postgres“. Kuna Docker Compose konfiguratsioonis

on konteinerid määratud samasse võrku piisab vaid nimepõhisest viitamisest et luua ühendus. [20]

Dokumenteerimaks serveris Docker konteinerite seadistust on Gitlab koodivaramus eraldi kaust server-configuration, mis dubleerib olulisemad seadistusfailid serverist koodivaramusse. Samuti on ka lokaalselt andmebaasi käivitamiseks local-configuration kaust, milles on docker-compose.yml koos posgres.env failiga.

```yaml
version: '3.7'
services:
  swag:
    container_name: swag
    image: linuxserver/swag:version-1.21.0
    cap_add:
      - NET_ADMIN
    networks:
      - apitransformations
    restart: unless-stopped
    env_file:
      - swag.env
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /root/data/swag:/config
      - /root/data/frontend:/srv/frontend
  postgres:
    container_name: postgres
    image: postgres:13.2-alpine
    networks:
      - apitransformations
    restart: always
    env_file:
      - postgres.env
    ports:
      - "5432:5432"
    volumes:
      - /root/data/postgres:/var/lib/postgresql/data
  nestjs:
    container_name: nestjs
    image: madispold/apitransformations:nestjs
    networks:
      - apitransformations
    restart: always
    env_file:
      - nestjs.env
    ports:
      - "8000:8000"

networks:
  apitransformations:
```

Joonis 8. Rakenduse Docker konfiguratsioon docker-compose.yml

5 Kasutajaliidese ja lahenduse kirjeldus

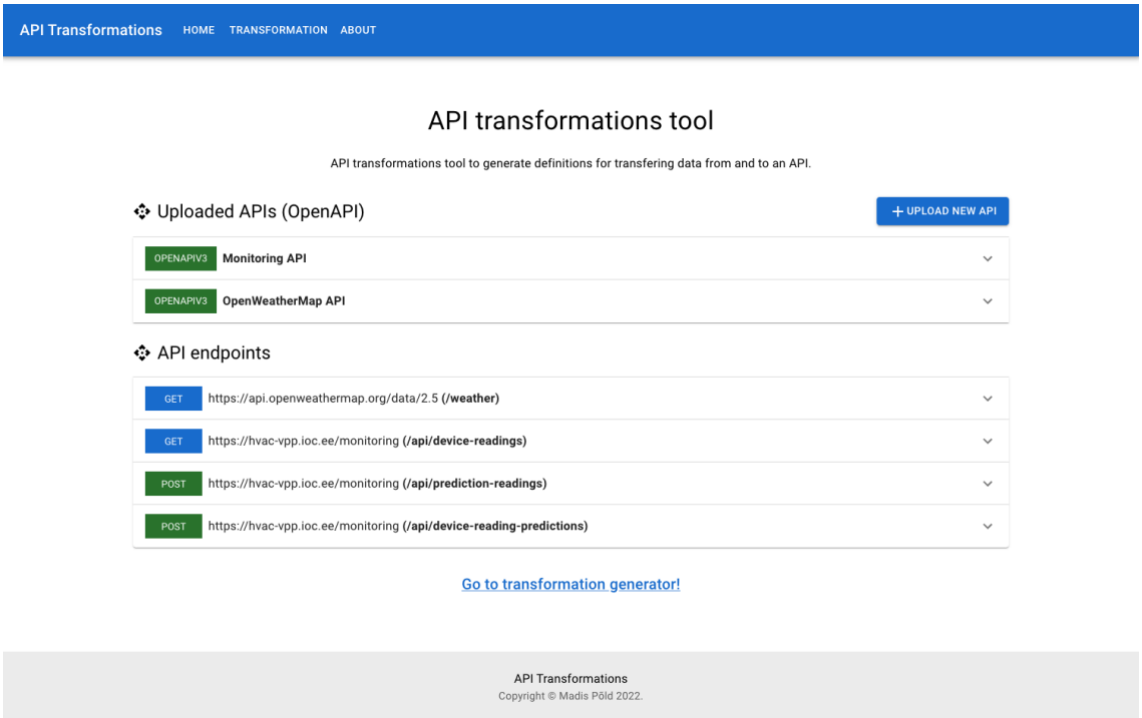
Antud peatükk annab ülevaate arendatud kasutajaliidesest ja sellega seotud teistest komponentidest koos realisatsiooni kirjeldusega. Rakenduse keel on inglise keel. Paigaldatud rakendus asub aadressil https://apitransformations.ee.

5.1 Kasutajaliides ja üldine disain

API teenuse üleslaadimisel MVP lahenduses on spetsifikatsioon limiteeritud OpenAPI peale ehk pole realiseeritud teistest formaatidest teenuse kirjelduse konverteerimist (e.g XML). [9]

Ühes andmete transformatsioonis on võimalik valida väliseid ehk allikana kasutatavaid API otspunkte ja ka sihtotspunkte kuni n tükki. Iga transformatsiooni protsess võimaldab luua seoseid kuni n allika või varasema transformatsiooni ja ühe sihtotspunkti vahel. Protsesse võib olla kuni n tükki. Allikana *(source)* valitud otspunktid on piiratud HTTP päringu tüüpidega GET ja POST. Sihtotspunkt *(target endpoint)* ei saa olla GET tüüpi ehk peab olema andmeid lisav või uuendav päring. [21]

Ülesehituselt on esitluskiht lihtne. Struktuurilt jaotuvad vaated kolmeks: päis, sisuosa ja jalus. ReactJS rakenduses on kõik kolm arendatud taaskasutatavate komponentidena. Navigeerimise järgselt laetakse uuesti vaid muutuvaid sisuosa komponente vastavalt rakenduses defineeritud asukohtadega kooskõlas URLidele. Igal vaatel on pealkiri ja kirjeldus ühtse struktuurina ja muu sisu on muutuv. Päis ja jalus on realiseeritud MUI komponentidega. Levinud komponendid lehel on MUI nupud, lingid, akordionid, modaalid ja sisestusväljad (Select, Input, Checkbox ja Textarea elemendid), mis on integreeritud ReactJS koodibaasiga. (Joonis 9) [15] [17]



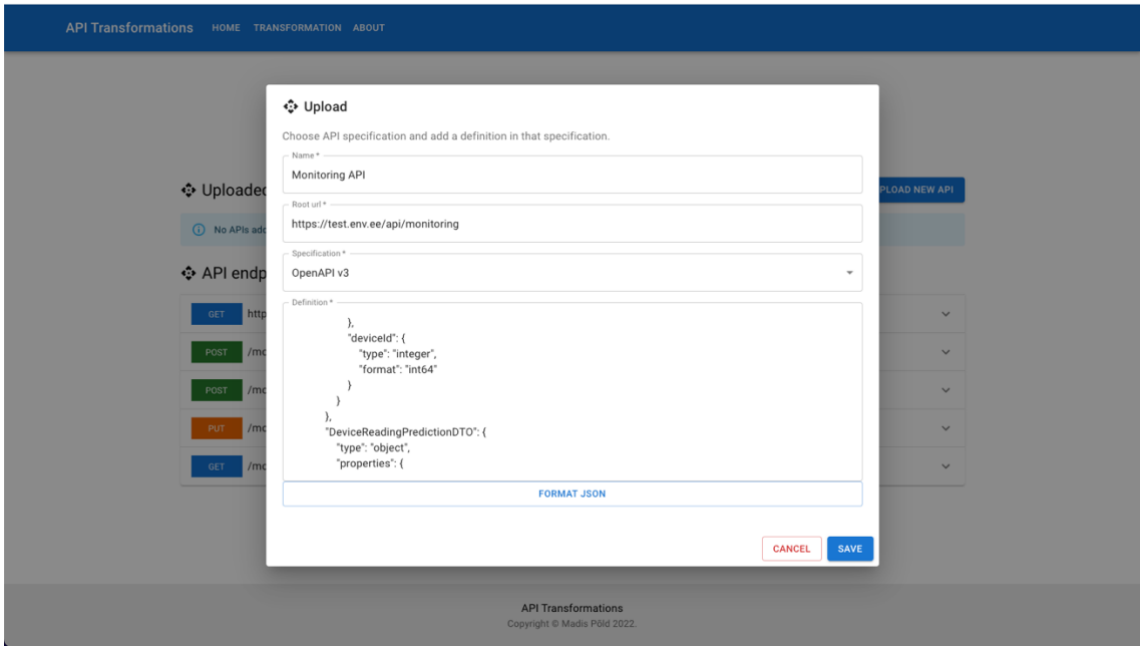
Joonis 9. Rakenduse avaleht.

5.2 Kindlas spetsifikatsioonis API teenuse üles laadimine

Avalehel on kaks akordioni plokki, milledest esimene on *Uploaded APIs* (üleslaetud API teenused) koos võimalusega laadida üles uus: vt. nupp *Upload new API*. Kui pole midagi üles laetud, siis kuvab rakendus *„No APIs added yet!“* infokasti. Üleslaetud teenuste andmestik salvestatakse NestJS backend rakenduse *Service* otspunkti POST päringuga. Erinevatel vaadetel päritakse vastavalt vajadusele salvestatud teenused GET päringuga samast otspunktist. [21]

Vajutades *Upload new API* nupule avatakse kasutajale modaal uue API teenuse lisamiseks. Modaal eeldab sisendina *Name* (nimi), teenuse domeeni *Root url* (absoluutne url), *Specification* (teenuse kirjelduskeel) ja *Definition* (valitud spetsifikatsioonis definitsiooni). Lisaks on võimalik kopeeritud definitsiooni vormindada abifunktsiooni abil *Format JSON* nupust. Nupu vajutusel tabuleeritakse sisendina antud JSON struktuur eeldusel, et see on standardile vastav.

Vormil (Joonis 10) on kohustuslikkuse kontrollid *Name* ja *Root Url* väljadel. Lisaks peab *Definition* välja sisend olema korrektne standardile vastav JSON, mida süsteem kontrollib ja annab kasutajale tagasisidet veateadete kaudu.

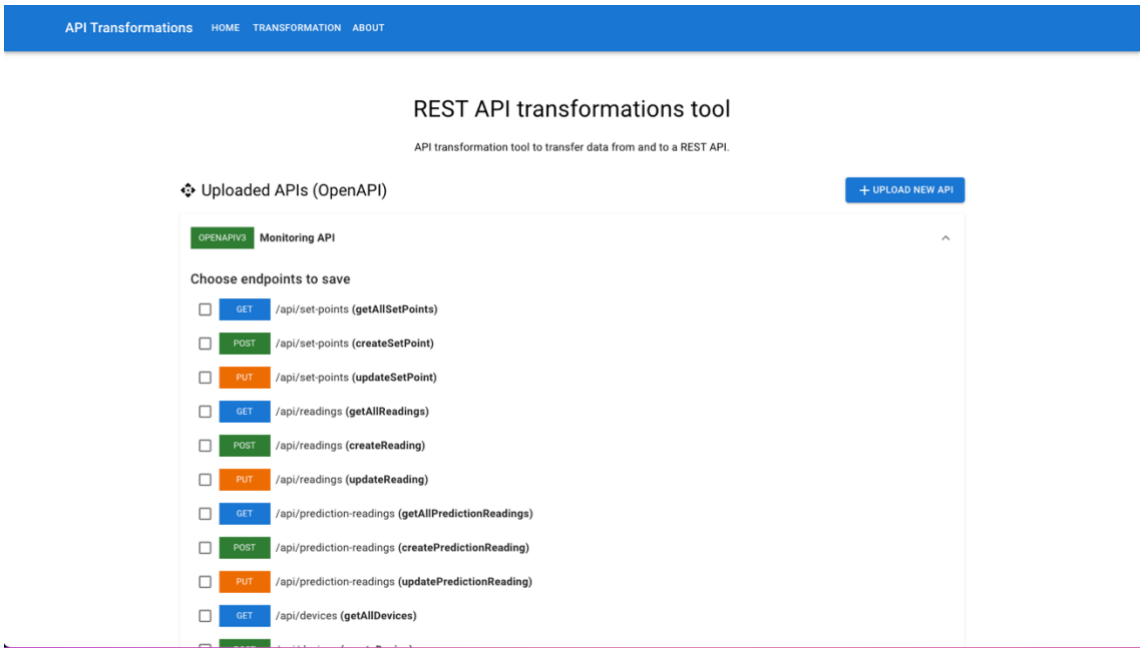


Joonis 10. Uue API teenuse lisamine.

Eduka salvestuse tulemusel modaal sulgub ja avalehele ilmub akordionisse lisatud teenus. Taustal aga kasutab rakendus *openapi-extract* teeki, et üleslaetud OpenAPI spetsifikatsioonist välja lugeda iga API otspunkt ja genereerida loetelust vorm otspunktide valimiseks ja salvestamiseks. [22]

5.3 API teenuse otspunktide vaatamine

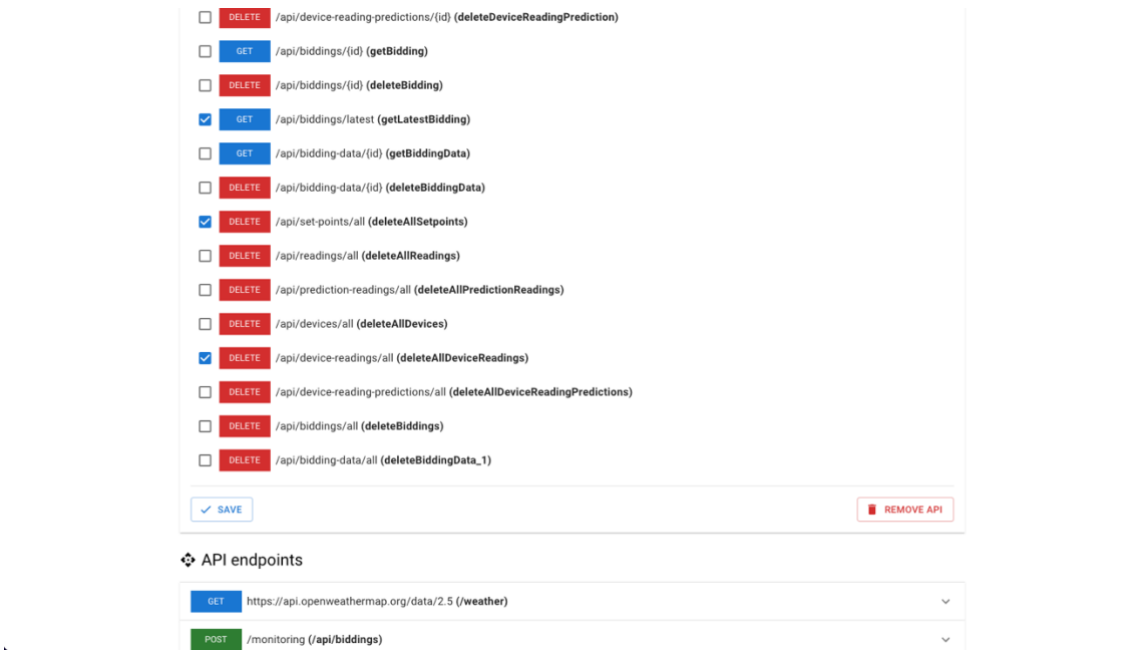
Kui süsteemi on edukalt üles laetud vähemalt üks API teenus, siis on võimalik avada konkreetne akordion ja vaadata seonduvaid otspunkte. Nimekirjas kuvatakse otspunkti HTTP päringu meetod, URL (teenuse asukoht) ja OpenAPI spetsifikatsioonist tulenev unikaalne identifikaator id. (Joonis 11) [21]



Joonis 11. API teenuse otspunktid.

5.4 API teenuse otspunktide salvestamine

Iga üles laetud API teenuse ja tema otspunktide kohta kuvab süsteem kirje ette valikruudu, mille täitmisel on võimalik valitud otspunktid süsteemi salvestada. Siinkohal salvestatakse otspunktid NestJS backend rakenduse Endpoint otspunkti POST päringuga. (Joonis 12)



Joonis 12. API teenuse otspunktide salvestamine.

Akordioni lõpus on *Save* (salvesta) nupp, mille vajutamise tulemusel käib rakendus iga valitud otspunkti kohta läbi tema struktuuri ja asendab „parameters“, „responses“ ja „requestBody“ objektidel viidad DTO skeemidele reaalsete skeemi objektidega.

See näitab ühte OpenAPI spetsifikatsiooni originaalkujul kasutamise puudujääki. On vaja teha lisatööd, et saada päringute kehandite struktuur kätte. Oma olemuselt on selline struktuur põhjendatud, kuna sama DTO võib olla mitme päringu vastuses ning nende otsene päringu külge lisamine tekitaks korduseid definitsioonis. Meie rakenduse arendamisel osutus see aga pigem takistuseks. Järgnevad kaks pilti illustreerivad asendusele eelnevat OpenAPI spetsifikatsiooni struktuuri ja sellele järgnevat tulemust. Eduka päringu tulemusel akordion suletakse ja täiendatakse *API Endpoints* nimekirja salvestatud kirjetega. (Joonis 13, Joonis 14) [9]

```
"responses": {
  "200": {
    "description": "OK",
    "content": {
      "*/*": {
        "schema": {
          "type": "array",
          "items": {
            "$ref": "#/components/schemas/ReadingDTO"
          }
        }
      }
    }
  }
}
```

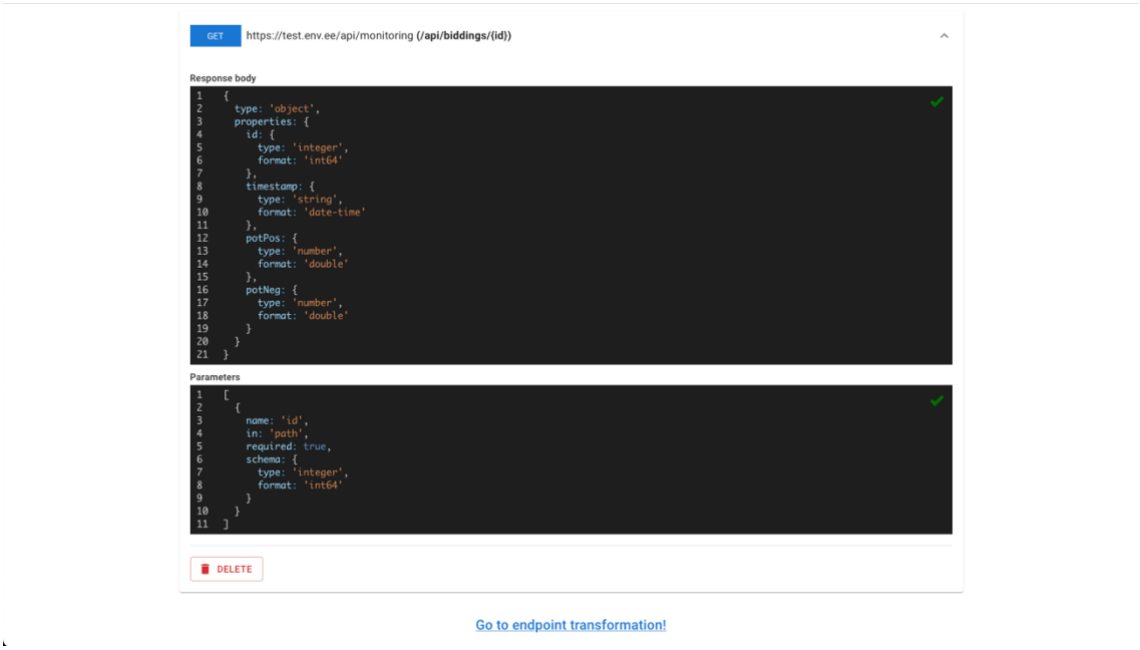
Joonis 13. API teenuse otspunkti salvestamine: algne kehand.

```
{
  type: 'array',
  items: {
    type: 'object',
    properties: {
      id: {
        type: 'integer',
        format: 'int64'
      },
      value: {
        type: 'number',
        format: 'double'
      },
      timestamp: {
        type: 'string',
        format: 'date-time'
      },
      deviceReadingId: {
        type: 'integer',
        format: 'int64'
      }
    }
  }
}
```

Joonis 14. API teenuse otspunkti salvestamine: lõplik kehand.

5.5 API teenuse otspunktide kustutamine

Kõik NestJS teenuse kaudu salvestatud API otspunktid kuvatakse *API Endpoints* akordionite nimekirjas. Iga otspunkti kohta kuvatakse nende olemasolul „Request Body“ „Response Body“ ja „Parameters“ objektid. Objektid kuvatakse *read-only* (muutmatus) režiimis spetsiaalses JSON sisestusväljas. Süsteem kuvab iga otspunkti kohta kustutamiseks *Delete* nupu. (Joonis 15) Kustutamise tegevuse käivitamisel sooritatakse ID põhine DELETE päring NestJS backend rakenduse Endpoint otspunktil ja eduka vastuse saamisel eemaldatakse nimekirjast kustutatud otspunkt.

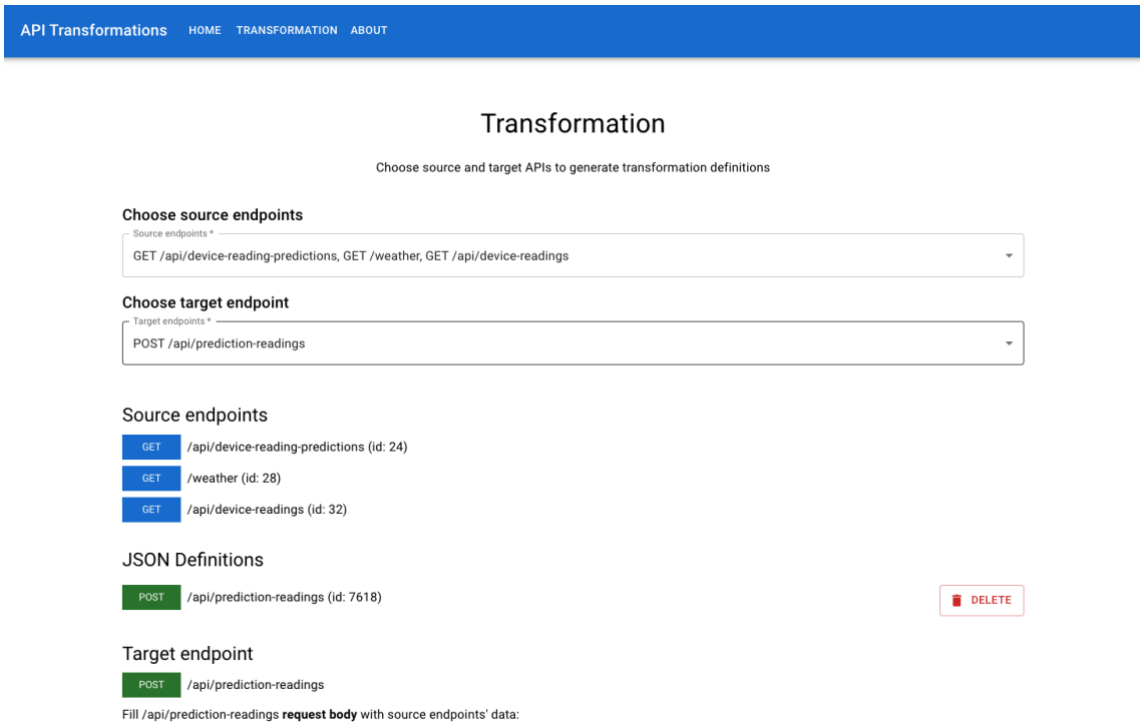


Joonis 15. API teenuse otspunkti vaatamine ja kustutamine.

5.6 Vastavusseoste loomine ja definitsioonide genereerimine

Olles süsteemi salvestanud vähemalt ühe GET või POST tüüpi päringu ja sellele lisaks vähemalt ühe mitte GET tüüpi päringu on võimalik alustada vastavusseoste loomist. Sinna saab menüüst *Transformations* lingiga või akordionite all avalehel oleva *Go to transformation generation* lingiga. (Joonis 15)

Andmete transformeerimise vaatel päritakse kõik salvestatud otspunktid NestJS backend teenuse Endpoint otspunktist ja kasutajal tuleb valida n arv *Source endpoints* (allikana kasutatavat otspunkti) ja n arv *Target endpoints* (sihtotspunkt). Iga sihtotspunkti kohta kuvatakse üks transformeerimise protsess. Kui mõlemasse sisestuskasti on tehtud vähemalt üks valik kuvatakse transformeerimise vorm. (Joonis 16).



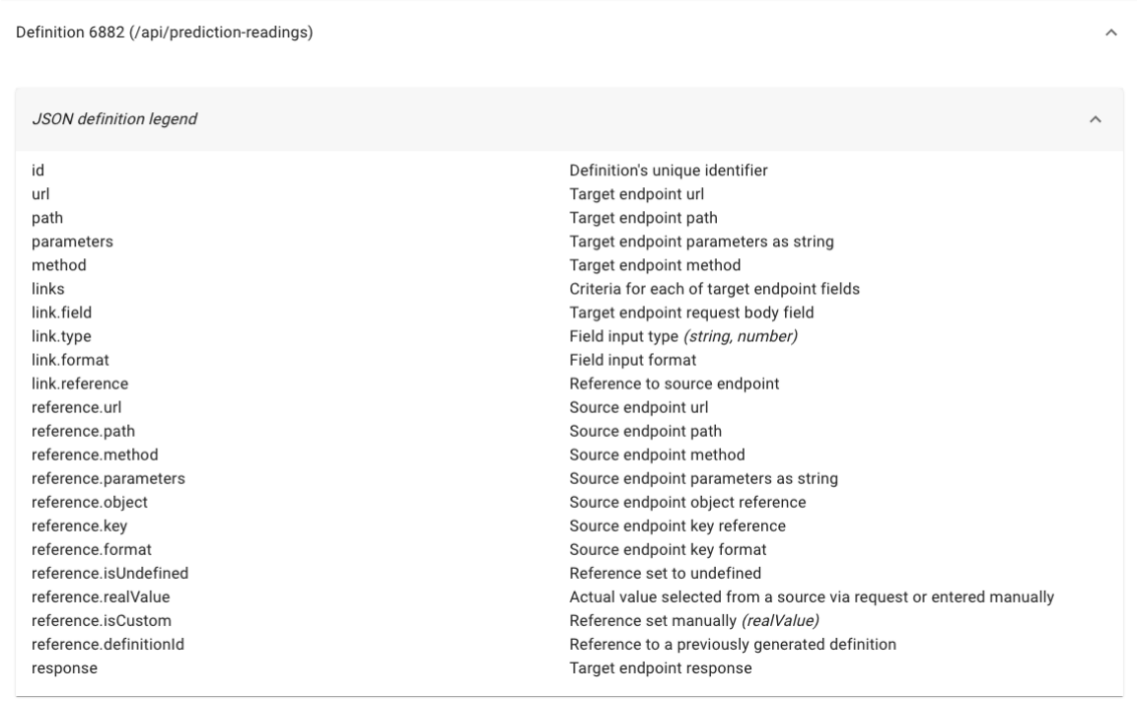
Joonis 16. Vastavusseoste loomine: transformatsiooni vormi päis.

Transformeerimise vormil kuvab süsteem:

- Allikate loetelu *Source endpoints* koos HTTP päringu meetodite ja URL viidetega.
- Salvestatud vastavusseoste definitsioonide loetelu *JSON Definitions* tingimusel, et neid on salvestatud protsessi raames.
- Sihtotspunkti kirjeldus *Target Endpoint* koos HTTP päringu meetodi ja URL viitega.
- Iga sihtotspunkti või tema koopia kohta üks vastavusseoste vorm (Joonis 20).
- Iga sihtotspunkti või tema koopia kohta üks vastavusseoste definitsioon koos seoste definitsiooni kirjeldava legendiga. Vastavusseoste vorme eraldab horisontaalne joon pealkirjaga *Next transformation.*

5.6.1 Vastavusseoste definitsioon

Vastavusseoste definitsioon kirjeldab vastavusseoste vormil loodud seoseid ja on mall või juhend seose põhjal reaalse päringu sooritamiseks. Rakenduses on definitsiooni kirjeldamiseks iga definitsiooni juurde kuvatud definitsiooni legend. (Joonis 17)



Joonis 17. Vastavusseoste loomine: definitsiooni legend.

Vastavusseoste definitsiooni unikaalne identifikaator on numbriline *Id* väli. *Links* massiiv kuvab iga sihtotspunkti päringu kehandis oleva välja kohta vormil valitud vaste ja viimast kirjeldava andmestiku. *Reference* objekt viitab allikale, millelt andmed võetakse, või reaalsele väärtusele, mis vormil sisestatud. *Object* ja *key* näitavad allika kehandist valitud vastet. *Object* võib puududa kui seos luuakse otse võtmega ja puudub objektiviit. *RealValue* on vormil sisestatud või päringu teel valitud reaalne lõplik väärtus sihtotspunkti salvestamisel tema kehandis. *isUndefined* määrab välja puuduvaks ja *isCustom* tähendab manuaalset sisendit *RealValue* külge. Igat vastavusseoste definitsiooni kirjeldavad järgmises tabelis loetletud väljad (Tabel 1).

Tabel 1. Vastavusseoste definitsiooni väljad.

Väli	**Kirjeldus**
id	Definitsiooni unikaalne identifikaator
url	Sihtotspunkti päringu üldine URL
path	Sihtotspunkti päringu täpsem viit
parameters	Sihtotspunkti päringu parameetrid
method	Sihtotspunkti päringu HTTP meetod
links (massiiv)	Seosed sihtotspunkti päringu iga välja kohta.
field [link]	Sihtotspunkti väli
type [link]	Sihtotspunkti välja sisestusviis (sõne või number)
format [link]	Sihtotspunkti välja formaat
reference [link] (objekt)	Allika päringu ja allika kehandist valitud välja kirjeldus
url [reference]	Allika päringu üldine URL
path [reference]	Allika päringu täpsem viit
parameters [reference]	Allika päringu parameetrid sõne kujul
object [reference]	Objektiviit allika päringu kehandis
key [reference]	Väljaviit allika päringu kehandis
format [reference]	Allika välja formaat
isUndefined [reference]	Väli määratud puuduvaks
realValue [reference]	Väljale määratud reaalne väärtus (mitte vastavus allikast)
isCustom [reference]	Väli manuaalselt sisestatud (mitte vastavus allikast)
definitionId [reference]	Viit definitsiooni päringu vastele (mitte vastavus allikast)
response	Sihtotspunkti päringu vaste struktuur

Näide eri tüüpi defineeritud seoste komplektist on järgnevatel piltidel (Joonis 18, Joonis 19), mis on genereeritud allpool (Joonis 20) kuvatud vastavusseoste vormilt.

Joonis 18. Vastavusseoste loomine: definitsioon.

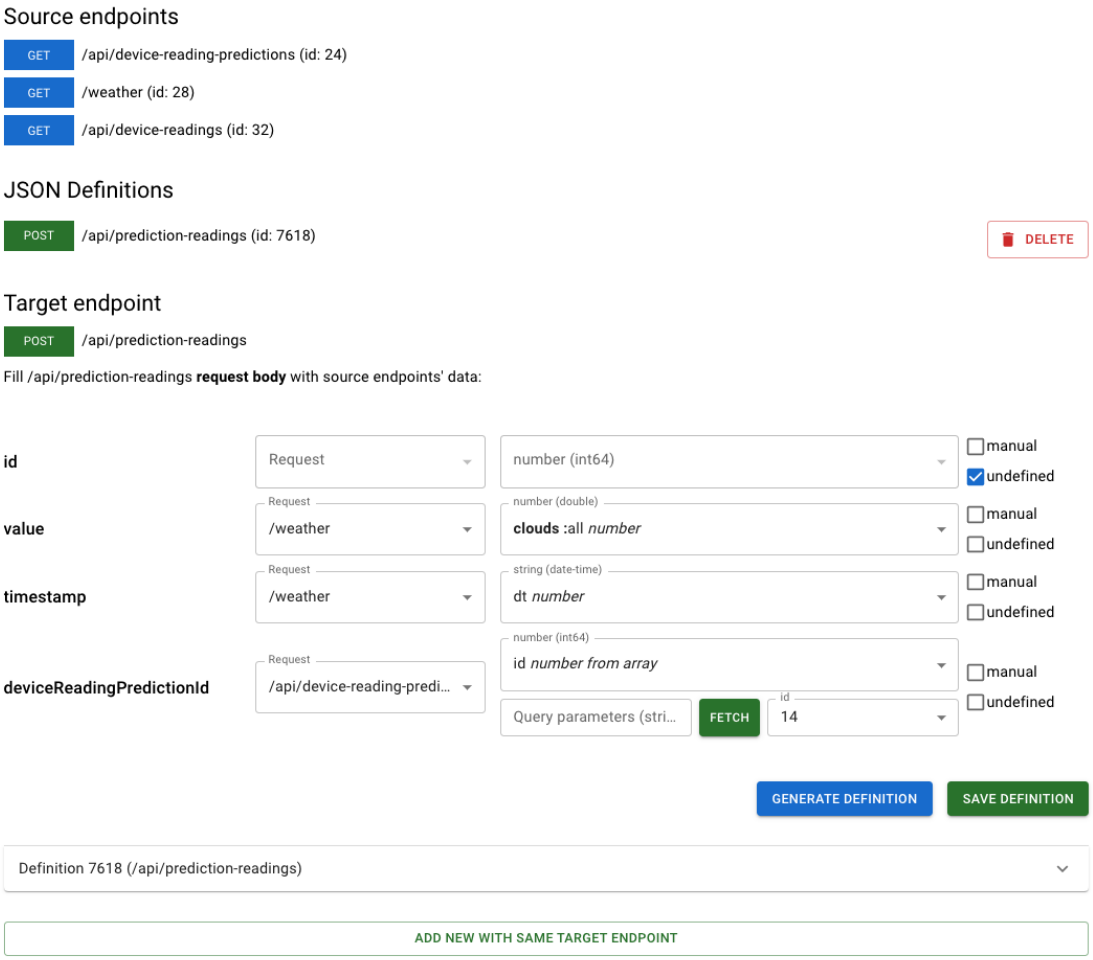
Joonis 19. Vastavusseoste loomine: definitsioon jätk.

5.6.2 Vastavusseoste vorm

Vastavusseoste vormil kuvatakse iga valitud sihtotspunkti kohta eraldi veeruna iga nõutud väli (nt. id*)* sihtotspunkti päringu kehandist. *Request* valikkastis kuvatakse allika valik ja sellel järgnevas valikkastis konkreetne väli päringust ehk vastavus. Vajadusel

saab määrata välja väärtuse *undefined* (defineerimata) linnutades samanimelise valikkasti. Seljuhul muutuvad sisestuskastid muutmatuks. Samuti saab sisestada vastavuse vabatekstina märgistades valiku *manual*, mille korral muutub valikkast hoopis sisestuskastiks. (Joonis 20)

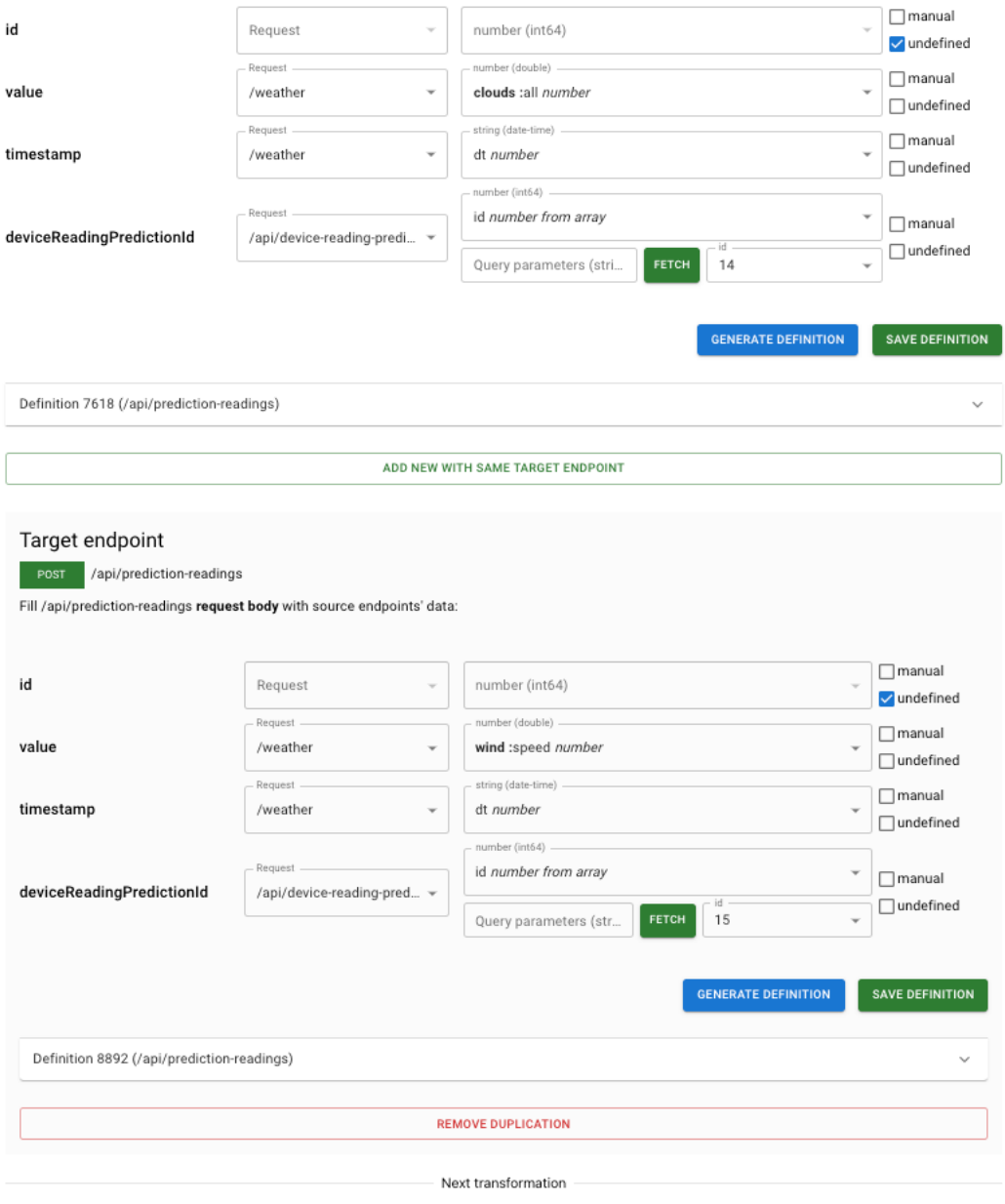
Kui valitud vastavuses tuvastatakse massiiviga päringu kehand, siis võimaldab süsteem andmete täiendava pärimise, et valida kehandist konkreetne väärtus. Päringu sooritamiseks kuvatakse välja valiku alla parameetrite sisestuskast ja nupp *Fetch*. Parameetrite lisamine pole kohustuslik. Nupu vajutusel sooritab süsteem päringu valitud otspunkti pihta ja kuvab valikkasti kõikvõimalikud väärtused konkreetsest väljast (nt. id). Päring sooritatakse läbi NestJS backend teenuse Proxy otspunkti Päringu järgselt on võimalik valida sobiv vastavus. (Joonis 20)



Joonis 20. Vastavusseoste loomine: vastavusseoste vorm.

5.6.3 Ühe sihtotspunkti kohta mitme vastavusseoste definitsiooni loomine

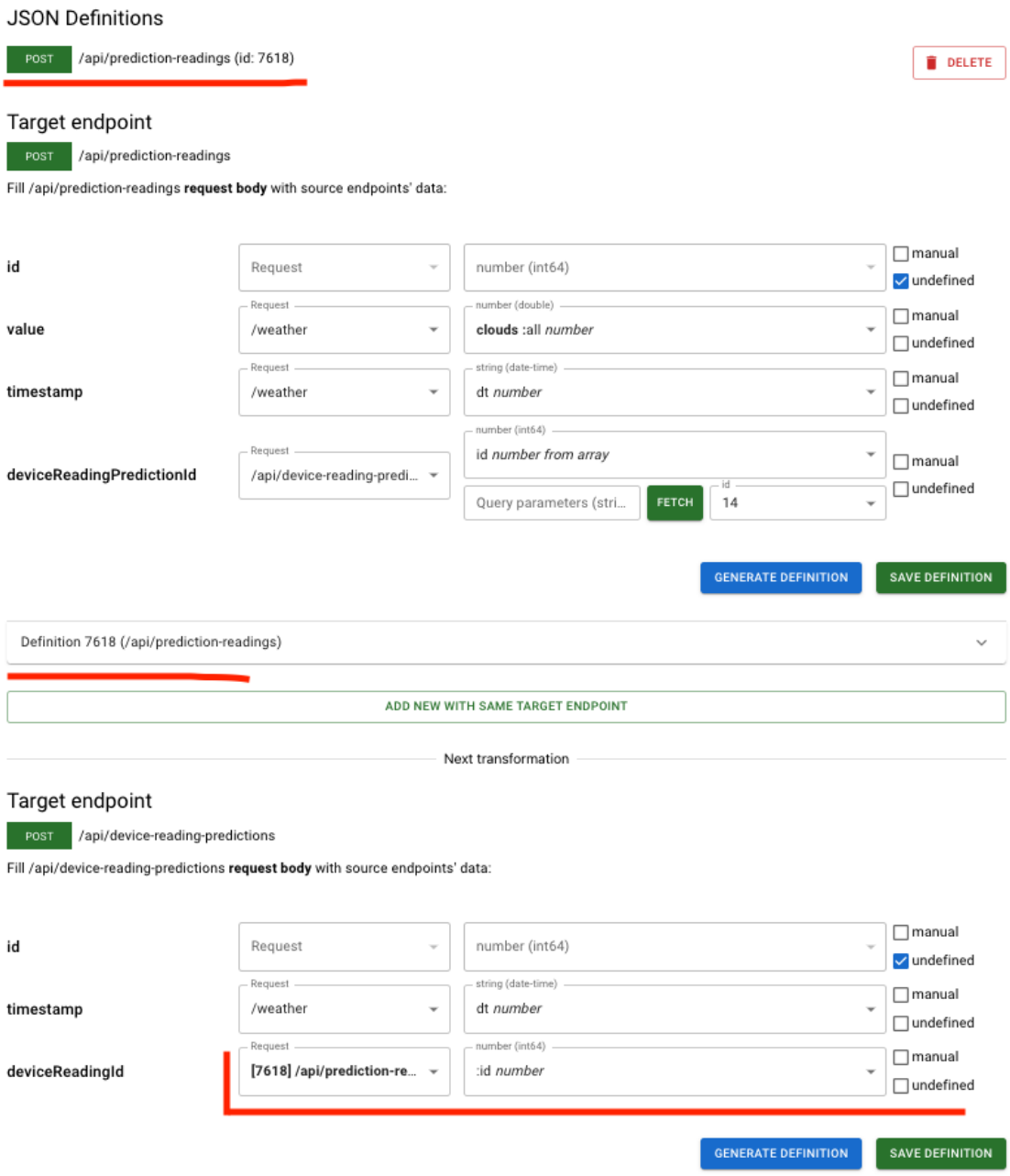
Rakenduses saab luua ühe sihtotspunkti kohta ka n arv koopiaid enda vastavusseoste vormi ja definitsiooniga. See võimaldab ühtsest allikast luua mitu sama põhjaga päringut paralleelselt. Koopia loomiseks on vaja vajutada nupul *Add new with same target endpoint* soovitud sihtotspunkti vastavusseoste vormi jaluses. Koopiad kuvatakse transformatsiooni vormil eristatava hallika taustaga ja neid on võimalik ka eemaldada nupust *Remove Duplication.* (Joonis 21)



Joonis 21. Vastavusseoste loomine: koopiad.

5.6.4 Vastavusseoste definitsioonide genereerimine ja salvestamine

Vormil loodud vastavusseoste põhjal saab genereerida definitsiooni ja lisaks edaspidiseks kasutamiseks definitsioon ka salvestada transformeerimise vormi kontekstis. Nupp *Generate definition* genereerib hetkel vormil aktiivsete valikute põhjal lõputöö raames loodud struktuuri (vt. peatükk 5.6.1) põhjal vastavusseosed. Olles genereerinud definitsiooni on see võimalik ka vormil kasutamiseks salvestada vajutades nupul *Save definition.* Seejärel lisandub definitsioon vormi päises olevasse salvestatud definitsioonide loetellu *JSON definitions* ja on kasutatav järgnevate sihtotspunktide vastavusseoste vormidel sisendina. Salvestatud definitsioonid ilmuvad teistel vastavusseoste vormidel *Request* valikkasti valikutena. Järgnev joonis kujutab olukorda, kus esimese vastavusseose tulemus ehk päringu vaste antakse sisendina järgmisesse vastavusseoste komplekti. (Joonis 22)



Joonis 22. Vastavusseoste loomine: definitsiooni tulemi kasutamine teise seose sisendina.

6 Testimine

Antud peatükis loome rakenduse abil seoseid ja genereerime definitsioonid, mille põhjal on võimalik sooritada reaalsed päringud, et imiteerida sissejuhatuses püstitatud ilmaandmete talletamise protsessi.

6.1 API teenuste otspunktide salvestamine

Esiteks lisame rakendusse kohaliku teenuse, milleks on Monitoring API. Monitoring API asub URLil https://hvac-vpp.ioc.ee/monitoring ja on saadaval OpenAPI v3 spetsifikatsioonis. Välise teenusena lisame varem mainitud OpenWeatherMap API, mis asub URLil https://api.openweathermap.org/data/2.5 ja selle kohta leidsin samuti definitsiooni OpenAPI v3 spetsifikatsioonis. [23]

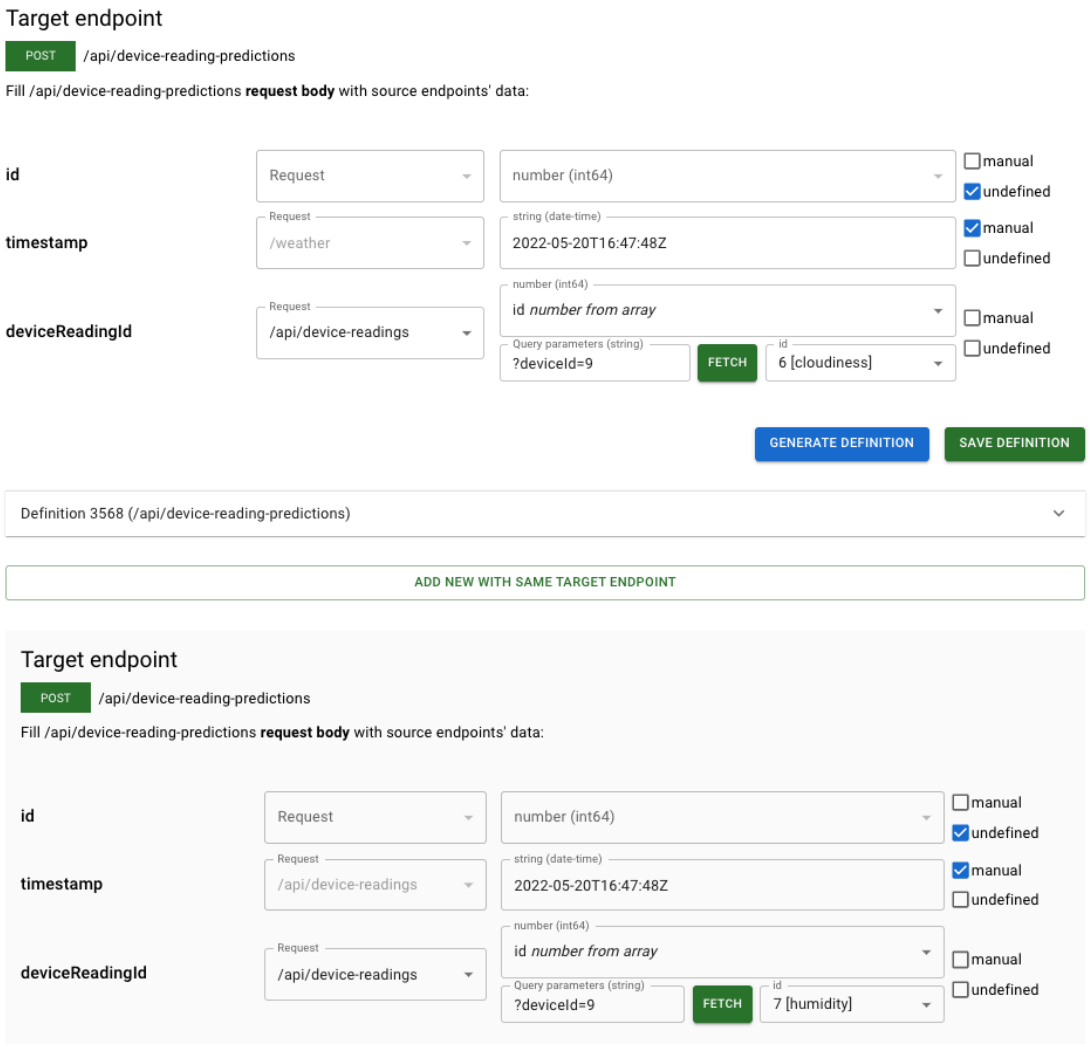
Monitoring APIst valime ja salvestame rakendusse API otspunktideks GET */api/device-readings (getAllDeviceReadings)*, mis tagastab konkreetse seadme kohta kõik mõõdikud (nt. niiskus, pilvisus, temperatuur jm.) ja POST */api/device-reading-predictions (createDeviceReadingPrediction)*, mis salvestab ennustuse kindlal ajahetkel konkreetse seadme mõõdiku (nt. niiskus) kohta ja POST */api/prediction-readings (createPredictionReading)*, kuhu salvestatakse eelnevas päringus talletatud ennustuse kohta reaalsed väärtused. OpenWeatherMap APIst salvestame rakendusse GET */weather (CurrentWeatherData)* API otspunkti, mis tagastab hetke ilma parameetrid asukoha või koordinaatide põhiselt. [24]

6.2 Vastavusseoste loomine ja definitsioonide genereerimine

Seejärel lähme vastavusseoste vormile ja lisame allikateks GET *CurrentWeatherData* ja GET *getAllDeviceReadings* otspunktid. Sihtotspunktideks määrame POST *createDeviceReadingPrediction ja* POST *createPredictionReading* otspunktid. Protsessi lihtsustamiseks kasutame juba süsteemi lisatud seadet (id: 9), millest saame vastavusseoste vormil sademete mõõdiku ja pilvisuse mõõdiku kasutades allikat *getAllDeviceReadings*.

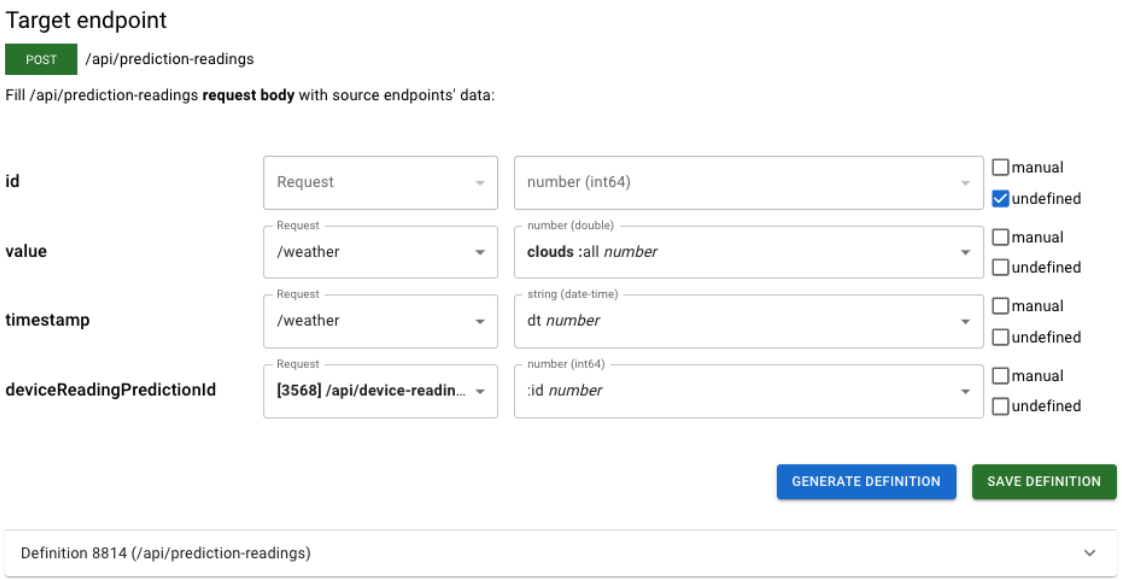
Nende mõõdikute kohta täidame kaks vastavusseoste vormi POST c*reateDeviceReadingPrediction* sihtotspunktile lisades ka ühe koopia. Id väärtus on

automaatselt genereeritud ja vastavuse määrame *undefined*. *Timestamp* väärtuse lisame protsessi lihtsustamiseks hetkel käsitsi ISO *date-time* formaadis ja *deviceReadingId* viitab vormides vastavalt pilvisusele ja niiskusele. (Joonis 23)



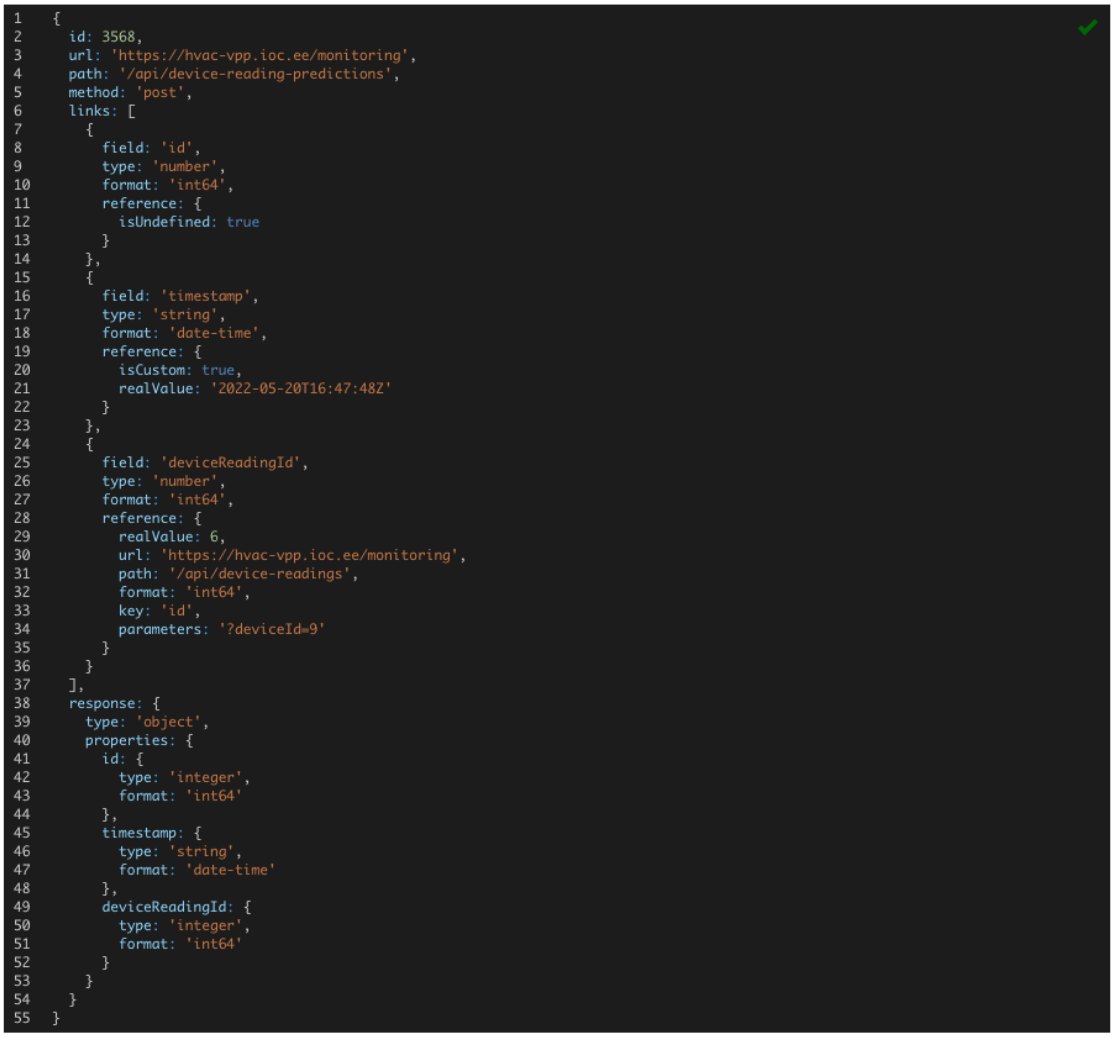
Joonis 23. Testimine: *createDeviceReadingPrediction* vastavusseosed.

Seejärel genereerime ja salvestame POST *createDeviceReadingPrediction* definitsioonid. Järgmisena loome POST *createPredictionReading* vastavusseoste vormi protsessi lihtsustamiseks hetkel vaid pilvisuse kohta. Id väärtuse määrame *undefined*, kuna see on automaatselt genereeritud. Reaalse ennustuse väärtuse võtame GET *CurrentWeatherData* päringust *clouds* objektist *all* võtmest. *Timestamp* väärtuse võtame sama päringu *dt* võtmest. Välja *deviceReadingPrediction* seome eelmise definitsiooni päringu tulemusel loodava *Id* väljaga. Lõpuks genereerime definitsiooni. (Joonis 24)

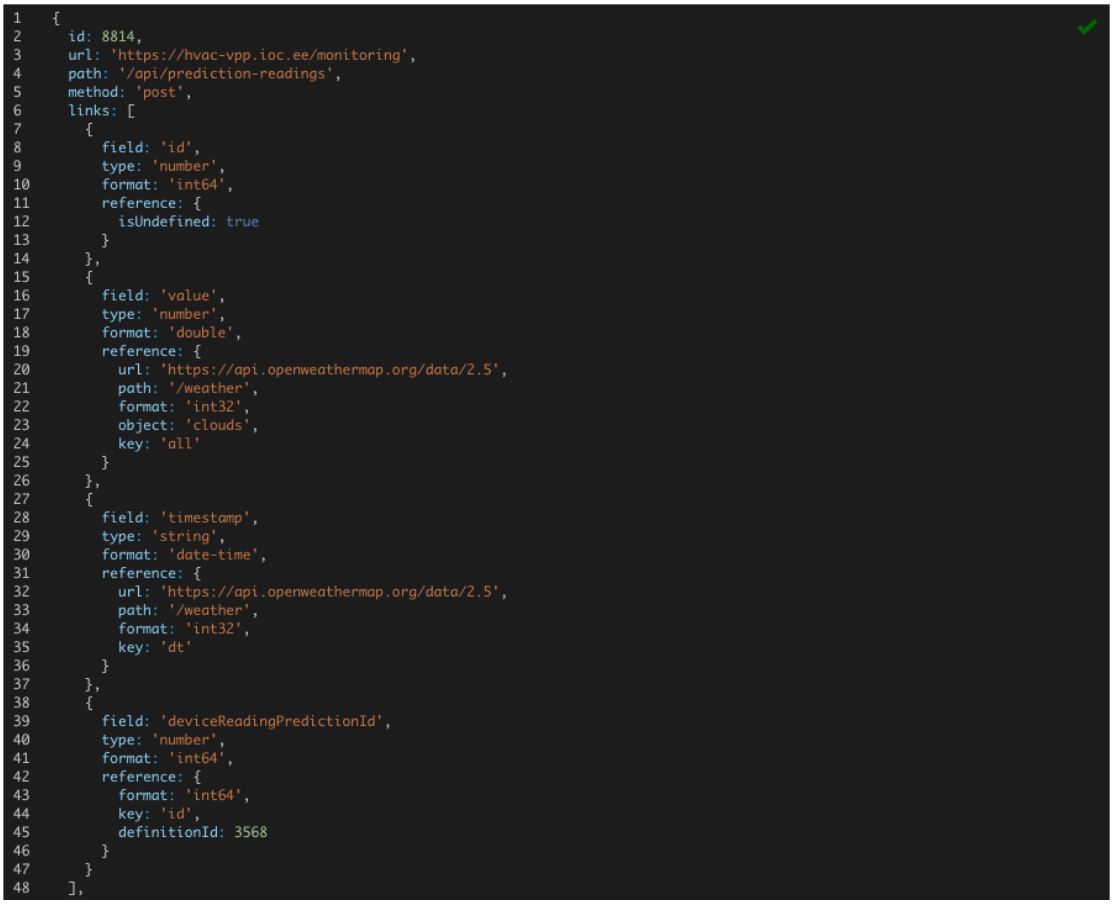


Joonis 24. Testimine: *createPredictionReading* vastavusseosed.

Selle protsessi tulemusel oleme sidunud seadme mõõturi ennustuse loomise ja loodud objekti külge ennustuste väärtuste salvestamise. Protsessi lihtsustamiseks sai kasutatud näites vaid hetke ilma väärtust. Genereeritud definitsioonidel on näha seost ID 3568 *createDeviceReadingPrediction* ja Id 8814 *createPredictionReading* vahel *definitionId* välja kaudu. Üks täiendav tegevus enne definitsioonide põhjal päringute sooritamist oleks kindlasti eri formaatides siht- ja allika otspunktide väljadel sooritada andmete teisendus. Näiteks Id 8814 *createPredictionReading* definitsioonis on selliseks väljaks *Timestamp*, mis saab allikast sisendi *int32* formaadis, aga nõuab väärtust *date-time* formaadis. (Joonis 25, Joonis 26)



Joonis 25. Testimine: *createDeviceReadingPrediction* definitsioon.



Joonis 26. Testimine: *createPredictionReading* definitsioon.

7 Kokkuvõte

Käesoleva lõputöö eesmärk oli genereerida kasutaja valitud APIde ja nende seoste põhjal JSON definitsioonid vastavusseostest, mille kaudu saab ühest või mitmest API-st liigutada ja seejuures vajadusel teisendada andmeid teise APIsse. Valminud rakenduses saab väga paindlikult luua vastavusseoseid n arv väliste ja sisemiste API otspunktide vahel ning genereerida nende kohta definitsioonid, mis sisaldavad kõike vajalikku päringute sooritamiseks. Definitsioon kirjeldab sihtotspunkti ja iga tema kehandis oleva välja kohta vastavuse allikale ja allika kehandis olevale väljale. Väljade kohta on defineeritud muuhulgas näiteks sisestustüüp, formaat, objektiviit või otseviit. Definitsioonis olev väljapõhine seos võib olla loodud ka hoopis varasema definitsiooniga ehk selle põhjal sooritatud päringu vastusega.

Põhjalikult on dokumenteeritud rakenduse eri komponendid koos rakenduse tarne ja arhitektuuriga. Süsteem on lihtsalt eri keskkondadesse paigaldatav ja laiendatav. Töö käigus oli palju õppetunde REST APIde ja nende spetsifikatsioonide ning eri standardite kohta. Üllatavaks kujunes OpenAPI spetsifikatsiooni omapärane ja keerukas struktuur. Täpsemalt päringu kehandi skeemide eraldi hoidmine muudest metaandmetest. Enamus töömahtu rakenduse arendusel kulus päringu struktuuride teisendamiseks sobilikku formaati, et kasutajaliides saaks neid tarbida ja oleks võimalik üksikutel otspunktidel tegevusi teostada.

Potentsiaalsete tuleviku täiendustena saaks realiseerida näiteks autoriseerimise ja kasutajapõhise päringute ning transformatsioonide ajaloo. Samuti ei võimalda praegune rakendus genereeritud definitsioonide alusel päringuid sooritada, mis annaks lahendusele suure lisaväärtuse. Lisaks piirab rakenduse paindlikkust OpenAPI spetsifikatsiooni nõue teenuste salvestamisel ja transformatsioonil kasutamisel.

Töö tulemusena valmis komponentide osas pea terviklik tarkvaralahendus, mis on hoiustatud avalikult domeenil apitransformations.ee. Esitluskiht on ehitatud ReactJS raamistikul kasutades MUI komponentide teeki. Esitluskiht suhtleb omakorda NestJS REST teenusega, et salvestada ja tarbida API teenuseid ja otspunkte ning sooritada päringuid välistesse teenustesse. NestJS rakenduse andmebaasina on kasutusel PostgreSQL DMS. Kogu lähtekood asub versioonihalduses Gitlab ja ReactJS rakenduse automatiseeritud uuendused on seadistatud Gitlab CI/CD kaudu. Server on renditud

Digitalocean pilveteenuspakkujalt ja rakendus jookseb Docker Compose konteinerlahendusena. [25]

Kasutatud kirjandus

[1] Stitch, „What is data transformation: definition, benefits, and uses,“ 01 04 2022. [Võrgumaterjal]. Available: https://www.stitchdata.com/resources/data-transformation.

[2] Microsoft, „Azure Data Factory: Copy and transform data from and to a REST endoint,“ Microsoft, 01 04 2022. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/azure/data-factory/connector-rest?tabs=data-factory.

[3] Microsoft, „Quickstart: Create a data factory by using the Azure portal and Azure Data Factory Studio,“ 15 01 2022. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/azure/data-factory/quickstart-create-data-factory-portal.

[4] Amazon Web Services, Inc., „AWS Data transformations for REST APIs,“ Amazon Web Services, Inc., 2022. [Võrgumaterjal]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/rest-api-data-transformations.html.

[5] Amazon Web Service, Inc., „AWS transformation: Photos example,“ Amazon Web Service, Inc., 2022. [Võrgumaterjal]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/example-photos.html.

[6] „DRY wikipedia,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Don%27t_repeat_yourself.

[7] „KISS wikipedia,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/KISS_principle.

[8] K. Ploesser, „Benefits of using the OpenAPI (Swagger) specification for your API?,“ Moesif, 15 01 2022. [Võrgumaterjal]. Available: https://www.moesif.com/blog/technical/api-design/Benefits-of-using-the-OpenAPI-Swagger-specification-for-your-API/.

[9] Swagger, „OpenAPI specification,“ [Võrgumaterjal]. Available: https://swagger.io/specification/.

[10] „PostgreSQL wikipedia,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/PostgreSQL.

[11] Altexsoft, „The Good and the Bad of TypeScript,“ Altexsoft, [Võrgumaterjal]. Available: https://www.altexsoft.com/blog/typescript-pros-and-cons/.

[12] K. Mysliwiec, „NestJS documentation,“ 2017. [Võrgumaterjal]. Available: https://docs.nestjs.com/.

[13] K. Mysliwiec, „NestJS CLI,“ 2017. [Võrgumaterjal]. Available: https://docs.nestjs.com/cli/overview.

[14] K. Myśliwiec, „NestJS database,“ [Võrgumaterjal]. Available: https://docs.nestjs.com/techniques/database.

[15] I. Meta Platforms, „ReactJS documentation,“ Meta Platforms, Inc., 2022. [Võrgumaterjal]. Available: https://reactjs.org/docs/getting-started.html.

[16] „Most used web frameworks among developers worldwide, as of 2021,“ Statista, [Võrgumaterjal]. Available:

https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/.

[17] Material UI SAS, „Material UI usage,“ Material UI SAS, [Võrgumaterjal]. Available: https://mui.com/material-ui/getting-started/usage/.

[18] Gitlab, „Gitlab CI/CD,“ [Võrgumaterjal]. Available: https://docs.gitlab.com/ee/ci/.

[19] D. Inc., „Docker,“ Docker, [Võrgumaterjal]. Available: https://www.docker.com/.

[20] D. Inc., „Docker Compose,“ Docker Inc., [Võrgumaterjal]. Available: https://docs.docker.com/compose/.

[21] Mozilla Corporation, „Http request methods,“ Mozilla Corporation, 03 10 2021. [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[22] M. Ralphson, „OpenAPI extract,“ [Võrgumaterjal]. Available: https://github.com/Mermade/openapi-extract.

[23] OpenWeather, „OpenWeatherMap API,“ 2012. [Võrgumaterjal]. Available: https://openweathermap.org/api.

[24] OpenWeather, „OpenWeatherMap API: Current weather,“ 2012. [Võrgumaterjal]. Available: https://openweathermap.org/current.

[25] M. Põld, „https://gitlab.cs.ttu.ee/Madis.Pold/iaib,“ 09 05 2022. [Võrgumaterjal]. Available: Thesis source code.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Madis Põld

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Interaktiivne veebivormi generaator kahe API vastavusseoste koostamiseks ja talletamiseks“ , mille juhendaja on Vahur Kotkas
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

09.05.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Appendix 1 – Non-exclusive license for reproduction and publication of a graduation thesis[1]

I, Madis Põld

1. grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis „Interactive Web Application to Generate Mapping Definitions for Data Transformations Between APIs“,

   supervised by Vahur Kotkas,

   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

09.05.2022

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.