KILP Arendusjuhend

Arenduses kasutatav tarkvara*

- Java 17 JDK
- Spring Boot 2.7.7
- PostgreSQL 13 + PostGIS
- Hibernate 5.6
- Spring Framework 5.3
- React 17
    - GUI (Javascript)
- React 18
    - tactical-ui (Typescript)
    - aviation-dashboard(Typescript)
- Lombok 1.18
- H2 integratsioonitestides 1.4
- Caffeine Cache 2.9
- Junit 5
- Mockito 4.5
- REST Assured 2.9
- Liquibase 4.9.0

**See info siin võib mingitel hetkedel olla mitte-ajakohane*

Rakenduse arhitektuur

- Rakenduste kood töötab serveris Java virtuaalmasinal, mille versioon on vähemalt 17.0
- Rakenduse konfigureerimisparameetrid on muudetavad väljaspool rakendust
- Rakendus toimib edasi võrgukatkestuse puhul
- Ühenduvust väliste liidestega saab kontrollida JMX vahenditega

Rakendus koosneb eraldiseisvatest valdkonnapõhistest moodulitest, mis on paigaldatavad iseseisvalt kas spring-boot rakendusena Maven-i pluginaga või Tomcati war-na. Rakenduse arendamisel järgitakse microteenuste arhitektuuri, mis võimaldab erinevatel meeskondadel rakendusi üksteisest sõltumatult arendada ja paigaldada.
Andmebaas on arendusfaasis kõikidele üks, asudes keskses serveris, millele on kõikidel arendajatel üle JDBC ligipääs. Arendajal on võimalik ka arendusmasina-lokaalset andmebaasi kasutada (H2 või PostgreSQL). Loodav lähtekood asub Git-is.

Rakenduse struktuur

Rakendus koosneb eraldiseisvatest moodulitest, mis suhtlevad omavahel üle REST-i. Mooduliteks jaotus on tehtud vastavalt valdkondadele ning moodulisisene struktuur on kirjeldatud järgnevalt:

Olemasolevad tehnilised lahendused ja joonised võivad kuuluda arenduse ja/või analüüsi käigus muutmisele. Kõik sisseviidavad muudatused peavad olema eelnevalt Tellijaga kooskõlastatud.

projektA
 src
  main
  resources
  java
   ee.smit.kilp.{rakenduse nimi}.{mooduli nimi}
   boot
   tests
   controllers
   services
   persistence
  test
   resources
   java

boot - spring-boot rakenduse käivitamiseks vajalik kood ja spring configuration klassid
tests - integratsioonitestid kogu rakenduse testimiseks
controllers - rest-i kontrollerid
services - teenused, mida kontrollerite kaudu välja kutsutakse
persistence - andmebaasiga suhtluseks vajalikud komponendid

Rakenduse konfiguratsiooniparameetrid

Rakenduse konfiguratsioon asub failis application.yml ning loetakse sisse rakenduse ülestulekul. Konfiguratsioon asub väljaspool rakendust ning parameetrid on muudetavad ilma rakendust restartimata, kasutades selleks spring-cloud-i funktsionaalsust.

Moodulitevaheline suhtlus

KILP omavaheliste moodulite suhtlus toimub üle:

- REST
- SSE
- RabbitMQ

Kõik moodulid, mis omavad REST liidest, väljastavad ka mooduli urli alt enda REST-i liidese spetsifikatsiooni, mida saab liidestuva mooduli arendaja kasutada liidestuse loomiseks.

Suhtlus KILP väliste süsteemidega

KILP väliste teiste andmekogude ja süsteemidega suheldakse kasutades:

- SOAP üle XTEE

- REST üle XTEE
- REST

Suhtlus andmebaasiga

Andmebaasiga suhtlusel kasutatakse jdbc datasource, mis on poolitavad. CRUD operatsioonide puhul kasutatakse Spring Data repository-sid, mis injectitakse vastavasse teenusklassi. Andmebaasi mudeli klassid luuakse andmebaasi pealt Hibernatetoolsi kasutades. Teenusloogika andmebaasi ei kirjutata ning loodav SQL funktsionaalsus püütakse hoida maksimaalselt andmebaasist sõltumatuna, sest see võimaldaks testimist käivitada H2 andmebaasi kasutades.

Puhverdamine

Puhverdamiseks kasutatakse Caffeine Cache'i.

Transaktsionaalsus

Transaktsionaalsust rakendatakse annotatsioonipõhiselt teenuskihis.

Logimine

Rakenduses kasutatakse slf4j API-t ning logbacki implementatsiooni. Loggeri lisamiseks klassile kasutatakse @slf4j annotatsiooni projektist Lombok. Logifail kirjutatakse failisüsteemi maha ning on masintöödeldav.
Iga päringu kohta tehakse vastavasisuline audit logi kirje.

Kasutajaliides

Kasutajaliides realiseeritakse eraldi javascripti/typescripti rakendusena, mis suhtleb serveris asuva teenuskihiga üle REST-i.

Testimine

Rakenduse testimiseks kasutatakse nii integratiooniteste kui ka unitteste. Integratsioonitestide jaoks luuakse moodulis alammoodul, kus asub testi käivitamiseks vajalik konfiguratsioon ning andmete eeltäitmine. Unittestid asuvad testitava koodiga samas moodulis.

Rakenduse monitoorimine

Monitoorimine toimub JMX-i vahenditega.

Rakenduse dokumentatsioon

Koodi kirjutamisel rakendatakse Robert C. Martin raamatus “Clean Code” kirjeldatud metodoloogiat ehk koodi kirjutatakse maksimaalselt isedokumenteeruvana. Vajadusel lisatakse kommentaarid JavaDoc-ina.

Koodi kvaliteet

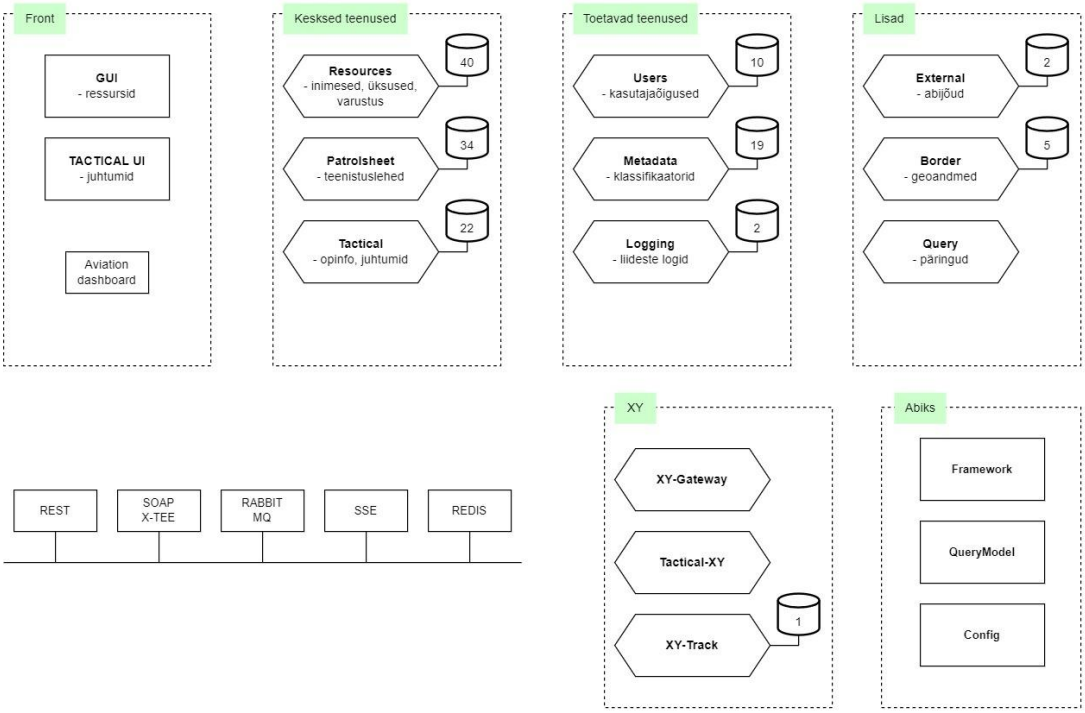
Koodi kvaliteedi formaalseks hindamiseks kasutatakse Sonar-it:

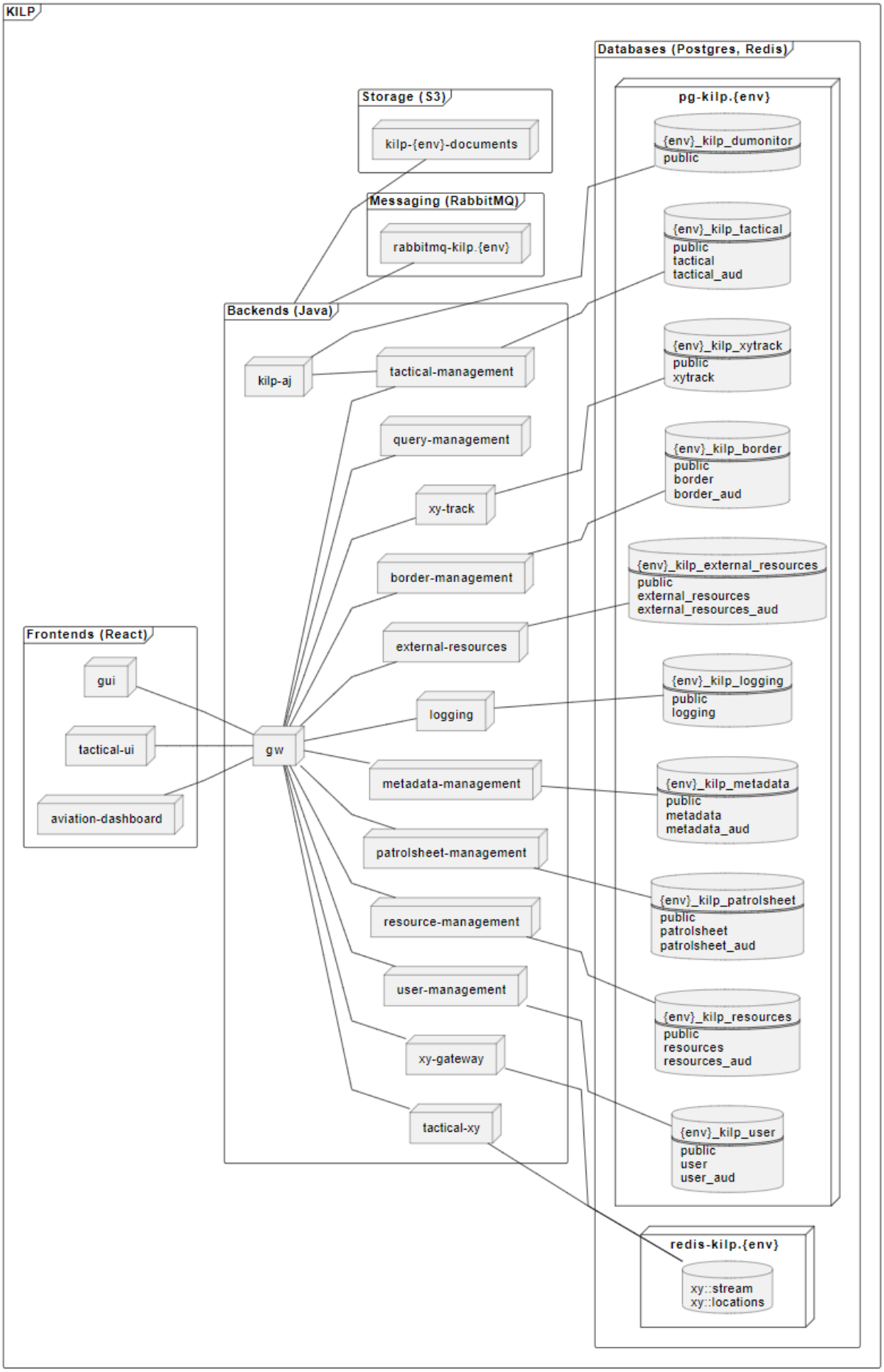
- Sonar-i hoiatused
- Automaattestidega kaetuse protsent (>=80%)

Mitteformaalne kvaliteet:

- Pull-requestide peer review'd.
- Koodi loetavus, lihtsus ja mõistetavus.
- Andmete töötluse kiirus ja optimaalsus.

KILP skeemid




KILP
Databases (Postgres, Redis)
pg-kilp.{env}
{env}_kilp_dumonitor
public
{env}_kilp_tactical
public
tactical
tactical_aud
{env}_kilp_xytrack
public
xytrack
{env}_kilp_border
public
border
border_aud
{env}_kilp_external_resources
public
external_resources
external_resources_aud
{env}_kilp_logging
public
logging
{env}_kilp_metadata
public
metadata
metadata_aud
{env}_kilp_patrolsheet
public
patrolsheet
patrolsheet_aud
{env}_kilp_resources
public
resources
resources_aud
{env}_kilp_user
public
user
user_aud
redis-kilp.{env}
xy::stream
xy::locations
Storage (S3)
kilp-{env}-documents
Messaging (RabbitMQ)
rabbitmq-kilp.{env}
Backends (Java)
kilp-aj
tactical-management
query-management
xy-track
border-management
external-resources
logging
gw
metadata-management
patrolsheet-management
resource-management
user-management
xy-gateway
tactical-xy
Frontends (React)
gui
tactical-ui
aviation-dashboard

KILP interfaces