DVK universaalkliendi API liidese loomine

# Eesmärk

Võtta kasutusel lahendus, mis lihtsustab DVK universaalkliendi liidestamist infosüsteemidega. Selleks on vaja luua API (Application Programming Interface), mis sisaldab DVK kliendi puhvertabelitega suhtlemise ja DVK XML konteineriga ümberkäimise jaoks vajalikke abiobjekte ja -funktsioone.

# Tehniline lahendus

DVKAPI põhineb programmeerimiskeelel *Java 1.6*. Andmebaasiobjektidele luuakse vastavad Java klassid, mis vastavad POJO[[1]](#footnote-1) mustrile. Need klassid seotakse andmebaasiga kasutades *ORM*[[2]](#footnote-2) raamistikku *Hibernate*[[3]](#footnote-3). See võimaldab arendajatel, kes liidestavad teisi süsteeme DVK-ga, lihtsamalt ja veavabamalt arendada kuna API-s on vajalikud abimeetodid ja klassid juba olemas. Samuti on võimalik sellisel juhul valutumalt teostada DVK klienditarkvara uuendamist, kuna DVK tarkvara uuendamisel on eeldatavalt võimalik suurem osa muudatusi kapseldada DVKAPIs ja kolmanda süsteemi DVK-ga liidestust ei ole vaja ümber programmeerida.

## Andmebaasioperatsioonid

Andmebaasiühenduste loomine on arvuti jaoks ajalises mõttes kallis protsess ning selleks, et andmebaasiühendusi ei peaks iga tegevuse jaoks uuesti looma, on olemas *Connection Pool*-id, kus hoitakse mitteaktiivseid andmebaasiühendusi. Antud API jaoks on samuti vaja luua lahendus, kus andmebaasiühendusi hoitakse *Connection Pool*-is.

API-s on funktsioon mis konfigureerib *Hibernate*-i ja tagastab *SessionFactory*. See *SessionFactory* hoiab endas viidet *ConnectionPool*-ile, mis haldab andmebaasiühendusi. Kui API-t kasutav programmikood pöördub näiteks objekti kustutamiseks API poole, siis antakse objektiga kaasa alati ka *SessionFactory* abil loodud sessioon. API kasutaja peab sellisel juhul hoolitsema *SessionFactory* säilitamise eest, sest vastasel juhul on vaja see uuesti luua ning seega luuakse ka uued andmebaasiühendused, mis on ressursinõudlik.

Koodinäide API poole pöördumisest:

public static void main(String[] args) {  
   
 SessionFactory sessionFactory = DVKAPI.createSessionFactory();

Honey honey = new Honey();  
 honey.setId(12);

DVKAPI.deleteHoney(honeyFactory.getCurrentSession());  
  
}

Koodinäide API funktsioonist, mis kustutab objekti:

private static void deleteHoney(Honey honey, Session session) {  
 Transaction tx = null;  
 try {  
 tx = session.beginTransaction();  
 session.delete(honey);  
 tx.commit();  
 } catch (RuntimeException e) {  
 if (tx != null && tx.isActive()) {  
 try {  
// Second try catch as the rollback could fail as well  
 tx.rollback();  
 } catch (HibernateException e1) {  
 logger.debug("Error rolling back transaction");  
 }  
// throw again the first exception  
 throw e;  
 }  
 }  
 }

Teine võimalus on määrata sessioon objekti muutujana (anda ette objekti loomisel või määrata vastavat meetodit kasutades):

public static void main(String[] args) {  
   
 SessionFactory sessionFactory = DVKAPI.createSessionFactory();

Honey honey = new Honey(sessionFactory.getCurrentSession());

// Saaks ka nii:

// Honey honey = new Honey();

// honey.setSession(sessionFactory.getCurrentSession());

honey.setId(12);

DVKAPI.deleteHoney(honey);  
  
}

private static void deleteHoney(Honey honey) {  
 Transaction tx = null;  
 try {  
 tx = this.getsession()**.**beginTransaction();  
 **this.getsession().**delete(honey);  
 tx.commit();  
 } catch (RuntimeException e) {  
 if (tx != null && tx.isActive()) {  
 try {  
// Second try catch as the rollback could fail as well  
 tx.rollback();  
 } catch (HibernateException e1) {  
 logger.debug("Error rolling back transaction");  
 }  
// throw again the first exception  
 throw e;  
 }  
 }  
 }

## Funktsioonide loetelu

Allpoololevate funktsioonide loetelu ei ole lõpik – s.t. töö käigus võib ilmneda et on vaja lisada funktsioone või et mõni siintoodud funktsioon ei ole vajalik.

**Message(dhl\_message)**

* *Getters/setters*
* getMessage(id)
* getMessageByIsIncoming(is\_incoming)
* getMessageBySenderOrgCode(sender\_org\_code)
* getMessageByRecipientOrgCode(recipient\_org\_code)
* getRecipients()
* addRecipient(Recipient)
* listMessages()
* saveMessage(Message)
* delete(id) – kustutab tabelist kirje (vajalik kui läks valesti vms).
* List<DataFile> getFiles() – võtab DATA väljal olevast väärtusest välja andmefailid.
* receiveMessages()
* sendMessages()
* checkStatuses()

**Recipient(dhl\_message\_recipient)**

* *getters/setters*
* getRecipient(recipient\_org\_code)
* getRecipientByPersonCode(recipient\_person\_code)
* getRecipientByOrgName(org\_name)
* getRecipientByName(recipient\_name)
* getRecipientBySendingDate(sending\_date)
* getRecipientBySendingDate(start\_date, end\_date)
* getRecipientByReceivedDate(received\_date)
* getRecipientByReceivedDate(start\_date, end\_date)
* getRecipientBySendingStatusID(sending\_status\_id)
* getRecipientByStatusID(recipient\_status\_id)
* getRecipientBy(fault\_code)
* …
* getMessage()
* listRecipients()
* saveRecipient(Recipient)
* delete(id)

**Settings(dhl\_settings)**

* *Getters/setters*
* getSettings(id)
* getSettingsByInstCode(institution\_code)
* getSettingsByPersonCode(personal\_id\_code)
* getSettingsByUnitID(unit\_id)
* void saveSettings(Settings)
* listSettings()
* delete(id)

**FolderSettings(dhl\_settings\_folders)**

* *Getters/setters*
* getFolderSettings(id)
* getFolderSettingsByFolderName(folder\_name)
* getFolderSettingsBySettingsID()
* listFolderSettings()
* saveFolderSettings(FolderSettings)
* delete(id)

**Organization(dhl\_organization)**

* *Getters/setters*
* getOrganization(org\_code)
* getOrganizationByOrgName(org\_name)
* getOrganizationsByDHLCapable()
* getOrganizationsByDHLDirectCapable()
* listOrganizations()
* saveOrganization(Organization)
* getOccupations()
* getSubdivisions()
* delete(id)
* checkOrganizations()

**Occupation(dhl\_occupation)**

* *getters/setters*
* getOccupation(occupation\_code)
* getOccupationByName(occupation\_name)
* getOccupationsByOrgCode(org\_code)
* listOccupations()
* saveOccupation(Occupation)
* delete(id)

**Subdivision(dhl\_subdivision)**

* *getters/setters*
* getSubdivision(subdivision\_code)
* getSubdivisionByName(subdivision\_name)
* getSubdivisionByOrgCode(org\_code)
* listSubdivisions()
* saveSubdivision(Subdivision)
* delete(id)

**Counter(dhl\_counter)**

* getValue()

**Test (testimiseks kasutatavad meetodid)**

## DVK Konteiner

DVKAPI hõlmab endas ka funktsionaalsust, mis võimaldab ümber käia DVK XML konteineriga. DVK konteineri jaoks olulised funktsioonid on XML konteineri loomine, selle omaduste muutmine ja andmebaasi salvestamine. Kuna DVK konteiner kapseldab endasse failid, adressaadid jne, siis on vajalikud ka funktsioonid nende andmete muutmiseks DVK konteineri sees. Eesmärk on kõrvaldada kolmandate infosüsteemidega liidestajatelt vajadus ise käsitsi kokku panna DVK konteinerit. Selle asemel on DVK API-s olemas vastavad klassid ja meetodid.

**Funktsioonide loetelu**

* *Konstruktorid* – tühja DVK konteineri loomiseks, eeltäidetud DVK konteineri loomiseks.
* *Getterid/setterid*
* toXML() – objektist XML strigi loomiseks
* writeToStream(stream) – objektist loodud XML stringi kirjutamine striimi
* generateGUID() – genereerib GUID (Globally Unique Identifier)

## Konfiguratsioonifailid

API koosseisus on ka konfiguratsioonifail, mis määrab ära kasutatava andmebaasi parameetrid. Siin tuleb mõelda, kas kasutada kindlaksmääratud nimega konfiguratsioonifaili või eeldada, et API kasutajad koostavad ise konfiguratsioonifaili (etteantud reeglite kohaselt) ning annavad parameetrina API-le selle faili asukoha/nime. Tegemist on Hibernate konfiguratsioonifailiga.

# Tööde loetelu

Nimekiri alamtöödest, mis antud ülesande juures on vaja teostada:

* Otsustada, kas antud tehnoloogiline lahendus on otstarbekas
* Hibernate konfiguratsioonifaili loomine
* Hibernate mapping faili koostamine vastavalt DVK kliendi puhvertabelitele.
* Andmetabelitele vastavate POJO objektide loomine (peaks saama genereerida vastava utiliidi abil)
* API Funktsioonide programmeerimine
* Juhend API kasutamiseks – tehniline kirjeldus, javadoc, näidisrakendus, näited API kasutamise kohta.

# Lisainfo

API programmeerimisel tuleb arvestada sellega, et DVK-s liiguvad tihti väga suured failid suurusjärgus 100MB. Nende failide töötlemiseks on parem kasutada striimist ja mitte hoida objekte mälus, kuna see sööb tohutult mälu ning muudab rakenduse aeglaseks.

1. POJO – Plain Old Java Object ( <http://en.wikipedia.org/wiki/Plain_Old_Java_Object> ) [↑](#footnote-ref-1)
2. ORM – Object-Relational Mapping (<http://en.wikipedia.org/wiki/Object-relational_mapping> ) [↑](#footnote-ref-2)
3. Hibernate ( <https://www.hibernate.org/> ) [↑](#footnote-ref-3)