1 Übersicht

Ein Test gilt als erfolgreich bestanden, wenn er alle Überprüfungen besteht, keine ungewollten Fehlermeldungen wirft und nicht vorzeitig abbricht.
Zudem erfüllen alle Unittests, die erfolgreich bestanden sind, die Bedingungen aller in den Sektionen gennanten Test-Frameworks für erfolgreiches Bestehen.

Der Test wurde erfolgreich bestanden	✓
Der Test wurde erfolgreich bestanden, allerdings nur unter der Vorraussetzung, dass benötigte andere Komponenten (z.B. Graphite, Datenbank) erreichbar sind	(✓)
Der Test ist fehlgeschlagen	X

2 Unittests

Für die Code-Überdeckung der folgenden tests, wurden JUnit 4.8.2 und kafka-streams-test-utils 1.1.0 verwendet.
Die verwendete Kafka-Version weicht minimal von der vorgegeben Testumgebung ab, da für die für uns vorgegebene Version keine kafka-streams-test-utils version existierte. Es wird Kafka 1.0.1 mit der Testumgebung für 1.1.0 verwendet.

2.1 ObservationDataDeserializer

Funktion	
serialisiertes Objekt deserialisieren	✓

2.2 GeoRectangle

Funktion	
anzahl an Sensoren (mit Einschränkung) auslesen	✓
gemittelte Sensordaten auslesen	✓
Sub-Polygon-Cluster generieren	✓
GeoJson des Polygons generieren (mit Wert zu einer Eigenschaft)	✓
prüfen ob ein Punkt enthalten ist	✓
alle SensorIDs zurückgeben lassen und überprüfen	✓

2.3 GeoRecRectangleGrid

Funktion	
einen Sensor zum Grid hinzufügen	✓
die enthaltenen Eigenschaften (z.B. temperature_celsius) überprüfen	✓
Daten des kompletten Grids an Graphite senden	(✓)
zwei Grids auf Equivalenz testen	✓

2.4 GridPropertiesFileManager

Funktion	
initialisierendes laden der Eigenschaften	✓

2.5 GradientPropertiesFileManager

Funktion	
initialisierendes laden der Eigenschaften	✓

2.6 KafkaPropertiesFileManager

laden und überprüfen der..

Funktion	
Dummy-Stream Eigenschaften	✓
Grid-Producer Eigenschaften	✓
Grid-Stream Eigenschaften	✓
Merge-Stream Eigenschaften	✓
Graphite-Stream Eigenschaften	✓
Graphite-Producer Eigenschaften	✓
Graphite-Connector Eigenschaften	✓
Export-Stream Eigenschaften	✓

2.7 WebServer

Funktion	
Server starten, dann in den Prozess eingreifen und ihn schließen	✓

2.8 ObservationDataToStorageProcessor

Funktion	
das mehrfache Einfügen von korrekten Daten in die Datenbank (zu unterschiedlichen Zeitpunkten) wird akzeptiert und der Zähler wird überprüft	(✓)
den Versuch, fehlerhafte Daten (aus falschen Parametern) zur Datenbank hinzufügen, stoppen	(✓)
den Versuch, korrekte Daten eines nicht existierenden GeoPolygons / Clusters zur Datenbank hinzufügen, stoppen	(✓)
bei fehlerhafter Verbindung zur Datenbank den Prozess des Kerns nicht gefährden	(✓)
Überprüfung vieler unterschiedlicher Zeitstempel auf Korrektheit und konvertierung in ein DateTime Objekt	✓
die Verbindung zur Datenbank kann neu aufgebaut werden	(✓)
es kann ein weiterer Memcached Server hinzufügt werden	(✓)

2.9 WebWorker

Server starten, eine Anfrage senden und überprüfen ob der Client eine gültige Antwort erhält..

Funktion	
eine Fehlerhafte Anfrage Senden und Bad-Request erhalten	✓
einen Gradienten abfragen	✓
alle Gradienten abfragen	✓
einen Sensor melden	✓
den Identifier eines GeoGrids herausfinden	✓
den Karten-Bereich eines GeoGrids herausfinden	✓
einen Sensor im GeoJson-Format abfragen	✓
ein odere mehrere GeoPolygone / Cluster im GeoJson-Format abfragen (bezogen auf live-Daten)	✓
ein odere mehrere GeoPolygone / Cluster im GeoJson-Format abfragen (bezogen auf Datenbank-Daten)(einzelner Zeitstempel)	(✓)
ein odere mehrere GeoPolygone / Cluster im GeoJson-Format abfragen (bezogen auf Datenbank-Daten)(mehrere Zeitstempel und Anzahl der Zeit-Schritte)	(✓)
alle bisher beobachteten Eigenschaften abfragen (z.B. temperature_celsius)	✓

2.10 GraphiteConnector

Funktion	
Daten von der Quelle entgegennehmen, Vorkehrungen Treffen und an die Konsole senden	✓

2.11 GraphiteSender

Funktion	
Daten vom Connector entgegennehmen, konvertieren und an Graphite senden	(✓)

2.12 ObservationData

Funktion	
toString konvertierung zu einem Json Objekt	✓

2.13 MultiGradient

Funktion	
Initialisierung mit null als Name, null als Farbwerte oder beidem	✓
Initialisierung mit gültigem Namen und einer einzelnen Farbe	✓
Initialisierung mit gültigem Namen und einer mehreren Farben	✓
zwei MultiGradients mit equals vergleichen	✓
die Farben als Gradienten in String form zurückgeben lassen	✓
eine GradientRange erstellen, hinzufügen, prüfen ob sie enthalten ist und sie entfernen	✓
einen SimpleGradient erstellen, hinzufügen, entfernen, ersetzen und in jedem Schritt die Farben kontrollieren	✓

3 Testszenarien

Hier werden sowohl alte als auch neue Szenarien getestet. Szenarien, die sich auf die im Pflichtenheft genannten Einträge beziehen werden durch die entsprechende Kennzeichnung (z.B. TI1000) deutlich gemacht.
Die Werte 1000 - 1070 deuten an, dass es sich um Server seitige Tests handelt.
Die Werte 2000 - 2100 deuten an, dass es sich um Client seitige Tests handelt.

3.1 Überarbeitete Szenarien

Szenario	Grund	Ersetzt durch
TI1000, TI1010, TI1020, TI1030, TI1040, TI1070	Die Konfigurations-GUI wurde durch Umgebungsvariablen ersetzt und die Prozesse beginnen mit dem Start des Docker-Containers (bzw. mit dem Ausführen der .jar Datei) automatisch. Hieraus ergibt sich, dass die Konfigurations-GUI und deren Einstellungsmöglichkeiten überflüssig geworden sind.	-
TI1050, TI1060	Aufgrund oben genannter Änderungen und wegen neuen Design-Entscheidungen wurde dieser Teil auf eine vom Kern seperat agierende Komponente verschoben - das Import- / Export- Servlet. Mit Hilfe dieses Servlets soll der Benutzer in die Lage versetzt werden, Daten vom Kern zu importieren und Daten zu exportieren.	QS2050, QS1060
TI2050	Dieses Szenario wurde mit einem anderen Szenario verschmolzen.	QS2050
-	Um das Servlet ohne Webinterface testen zu können, wurde ein neues Szenario erstellt.	QS2100

3.2 QS1060 - Import von Daten mit dem Servlet

Voraussetzungen

- FROST-Server
- Die Adresse des FROST-Servers muss bekannt sein und auf "/v1.0/" enden
- Importer (pim.jar) muss vorhanden sein
- Testdateien müssen vorhanden sein (test.csv, multi1test.csv und multi2test.csv)

Ablauf

1. Starten der pim.jar per Doppelklick oder über die Kommandozeile
2. Folgende UI-Elemente müssen in der geöffneten Anwendung zu sehen sein:
   - Zwei Textfelder, das eine mit einem Link, das andere als Text
   - Zwei Knöpfe, der eine zum auswählen von Dateien, der andere zum Importieren
   - Eine leere Tabelle
3. Im oberen Textfeld mit dem Link wird der bekannte Link zum FROST-Server eingefügt werden
4. Im unteren Textfeld wird der Zusatz für die IoT-IDs eingetragen, der die Daten dieses Imports eindeutig kennzeichnet
5. Als nächstes wird der Knopf mit der Aufschrift "Choose Files" gedrückt
6. Es öffnet sich ein Auswahlfenster
7. Nun werden die Testdaten ausgewählt
   - Variante 1: nur test.csv
   - Variante 2: multi1test.csv und multi2test.csv
8. Alle ausgewählten Dateien werden in der Tabelle aufgelistet und der Fortschritt beträgt 0%
9. Nun wird der Knopf mit der Aufschrift "Import Files" gedrückt
10. Anschließend startet der Import und es folgt keine Fehlermeldung
11. Es wird bis zu 5 Minuten gewartet oder bis 100% Fortschritt erreicht sind
12. Ist der Fortschritt nach dieser Zeit noch nicht bei 100%, so gilt der Test als nicht bestanden

13. Zur Überprüfung der Daten wird der bekannte Link zum FROST-Server aufgerufen und die Endungen: Sensor, ObservedProperties, Things, FeaturesOfInterest, Datastreams und Observations nach einanander angefügt (den Vorgänger wieder löschend)
    - Variante 1: Es ist genau ein Eintrage für jede Endung vorhanden (ein Eintrag ist hier ein json mit einer IoT-ID)
    - Variante 2: Es sind genau zwei Einträge für jedes vorhanden
    - Die angezeigten IoT-IDs müssen mit dem Inhalt des zweiten angegebenen Texfeldes anfangen
14. Zur genauen Prüfung werden die csv-Dateien geöffnet
15. Jede in dem csv enthaltene Information muss auch in Frost vorhanden sein, wobei der erste Eintrag in jeder Zeile gibt immer an, um welchen Typ es sich handelt.

3.3 TI2000 - Darstellung des berechneten Mittelwertes

3.4 TI2010 - Darstellung von Clustern ohne Sensoren

3.5 TI2020 - Darstellung der Tabellarischen Ansicht

3.6 TI2030 - Benutzen mehrerer Webinterface-Instanzen

3.7 TI2040 - Auswahl eines Sensortyps

3.8 QS2050 - Export von Daten mit dem Servlet

Voraussetzungen

- FROST-Server
- Kafka-System
- Pavos-Bridge zwischen FROST und Kafka
- Pavos-Core
- Export-Docker
- Pavos-Webinterface
- Ein abgeschlossener Import-Test auf diesem System
- Keine weiteren eingespeisten Daten auf dem System

Ablauf

1. Das Webinterface wird im Browser geöffnet
2. Auf der Karte wird ein Cluster ausgewählt, das Deutschland enthält
3. Der Knopf mit der Aufschrift "Export"
4. Es öffnet sich ein Overlay mit dem Titel Export in das folgendes eingetragen wird:
   - Im Feld "From" wird das Datum von vor einem Monat eingetragen
   - Im Feld "To" wird das aktuelle Datum eingetragen
   - Im Menüpunkt "Choose Clusters" wird "All Selected" ausgewählt
   - In der Kategorie "Choose Sensortype" wird "temperature_celsius" ausgewählt
   - In der Kategorie "Choose Exportformat" wird "CSV" ausgewählt
   - Es wird auf den Knopf mit der Aufschrift "Apply" geklickt
   - Die Exportierten Daten werden nach spätestens 5 Minuten heruntergeladen

3.9 TI2060 - Wiederholungen anzeigen

3.10 TI2070 - Echtzeitdarstellung

3.11 TI2080 - Detailansicht eines Sensors

3.12 TI2090 - Melden eines Sensors

3.13 QS2100 - Direktes Ansprechen des Servlets

Voraussetzungen

- FROST-Server
- Kafka
- Pavos-Bridge zwischen FROST und Kafka
- Pavos-Core mit Daten gefüttert
- Export-Docker
- Der Link zum Export-Servlet muss bekannt sein (endet auf "/edms/get?" vor der Angabe von Parametern). Dieser Link wird hier Beispielhaft als SERVLET? gekürzt.

Ablauf

Test 1 - Erlaubte Dateiendungen abfragen

1. Der Link "SERVLET?requestType=getExtensions" wird im Browser aufgerufen
2. Die Webseite enthält nur Text
3. Dieser Text ist eine mit Komma separierte Liste aller erlaubter Dateiendungen für den Export der Daten
4. Die Liste enthält mindestens einen Eintrag (normalerweise csv)

Test 2 - Nicht vorhandene Datei

1. Der Link "SERVLET?requestType=getStatus& downloadID=test" wird im Browser aufgerufen
2. Die Webseite enthält nur Text
3. Der Text ist "noID"
4. Der Link "SERVLET?requestType=tryDownload& downloadID=test" wird im Browser aufgerufen
5. Es wird eine Http-Fehlermeldung zurückgegeben

Test 3 - Datei exportieren Hier muss bekannt sein, welche Messgrößen (z.B. "temperature_celsius"), welche erlaubten Export-Dateiendungen und welche Cluster (ID muss bekannt sein) existieren.

1. Es wird aus den bekannten Messgrößen, Export-Dateiendungen und Clustern jeweils eins ausgewählt
2. Der Link "SERVLET?requestType=newExport& downloadID=test& extension=FORMAT& timeFrame= DATES:ssZ& observedProperties= OBSERVEDPROPERTIES& clusters=CLUSTER"

   Wobei FORMAT gleich der ausgewählten Export-Dateiendung entspricht. DATES, wird durch einen Datums-Bereich der zu importierenden Sensor-Werte im Format YYYY-MM-DDThh:mm:ssZ mit Komma getrennt (und mit maximal zwei Daten) ersetzt. OBSERVEDPROPERTIES entspricht der ausgesuchten Messgröße und CLUSTER dem ausgesuchten Cluster.
3. Es wird "started" als Text zurückgegeben
4. Der Link wird ein zweites mal aufgerufen
5. Es wird "duplicate" als Text zurückgegeben
6. Der Link "Servlet?requestType=getStatus& downloadID=test" wird im Browser aufgerufen

7. Die Webseite enthält nur Text
8. Dieser Text ist entweder "false" oder "true"
9. Wird "false" zurückgegeben, muss solange gewartet oder neu geladen werden, bis "true" zurückgegeben wird.
   Die Wartezeit skaliert mit der Größe der Anfrage und sollte bei kleinen Größen 10 Minuten betragen.
10. Sobald "true" zurückgegeben wird, wird automatisch eine Datei namens "test.zip" heruntergeladen, die die exportierten Daten enthält.