ДОДАТОК В

ТЕКСТ ПРОГРАМИ

```
import com.epam.dlex.repository.ComparisonConfigRepository;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.model.Filters;
import com.mongodb.client.result.UpdateResult;
import org.bson.Document;
import org.bson.conversions.Bson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Arrays;

import static com.epam.dlex.repository.imp.mongo.MongoConstants.*;

/**
 * @author Dmytro_Petrenko
 */
public class MongoComparisonConfigRepositoryImpl implements
ComparisonConfigRepository {

    private final MongoCollection<Document> mongoCollection;
    private static final Logger LOG =
LoggerFactory.getLogger(MongoComparisonConfigRepositoryImpl.class);


    public
MongoComparisonConfigRepositoryImpl(MongoCollection<Document>
mongoCollection) {
        this.mongoCollection = mongoCollection;
    }

    @Override
    public boolean create(String clientName, String newValue) {
        Document newDocumentState =
Document.parse(String.format(SAVE_COMPARISON_CONFIG, newValue));
        Bson filter = Filters.eq(CLIENT_FIELD_NAME, clientName);

        UpdateResult result = mongoCollection.updateOne(filter,
newDocumentState);
        return result.getModifiedCount() > 0;
    }

    @Override
    public boolean deleteRuleSequence(String clientName) {
        Document newDocumentState =
Document.parse(DELETE_RULE_SEQUENCE);
        Bson filter = Filters.eq(CLIENT_FIELD_NAME, clientName);
        UpdateResult result = mongoCollection.updateMany(filter,
newDocumentState);
        return result.getModifiedCount() > 0;
    }

    @Override
    public boolean update(String clientName, String newValue) {
        return create(clientName, newValue);
    }

    @Override
    public boolean delete(String clientName) {
        Document newDocumentState =
Document.parse(DELETE_COMPARISON_CONFIG);
        Bson filter = Filters.eq(CLIENT_FIELD_NAME, clientName);
        UpdateResult result = mongoCollection.updateOne(filter,
newDocumentState);
        return result.getModifiedCount() > 0;
```

```
    }

    @Override
    public String read(String clientName) {
        Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, clientName));
        Document project = Document.parse(PROJECT_COMPARISON_CONFIG);
        Document result =
mongoCollection.aggregate(Arrays.asList(matchClient,
project)).first();
        if (result == null) {
            return null;
        }

        Document value = (Document)
result.get(COMPARISON_CONFIG_FIELD_NAME);
        return value == null || value.isEmpty() ? null :
value.toJson();
    }

    @Override
    public boolean create(String clientName, String newValue, int
step) {
        Document newDocumentState =
Document.parse(String.format(SAVE_COMPARISON_CONFIG_BY_STEP,
newValue));
        Bson filter = Filters.and(Filters.eq(CLIENT_FIELD_NAME,
clientName), Filters.elemMatch(DATA_FIELD_NAME,

Document.parse(String.format(SAVE_COMPARISON_CONFIG_BY_STEP_CONDITION,
step))));
        UpdateResult result = mongoCollection.updateOne(filter,
newDocumentState);
        return result.getModifiedCount() > 0;
    }

    @Override
    public boolean update(String clientName, String newValue, int
step) {
        return create(clientName, newValue, step);
    }

    @Override
    public boolean delete(String clientName, int step) {
        Document newDocumentState =
Document.parse(DELETE_COMPARISON_CONFIG_BY_STEP);
        Bson filter = Filters.and(Filters.eq(CLIENT_FIELD_NAME,
clientName),
                Filters.elemMatch(DATA_FIELD_NAME,
Document.parse(String.format(SAVE_COMPARISON_CONFIG_BY_STEP_CONDITION,
step))));
        UpdateResult result = mongoCollection.updateOne(filter,
newDocumentState);
        return result.getModifiedCount() > 0;
    }

    @Override
    public String read(String clientName, int step) {
        Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, clientName));
        Document matchIndex =
Document.parse(String.format(MATCH_INDEX_REQUEST, step));
        Document unwind = Document.parse(UNWIND_REQUEST);
```

```
        Document project =
Document.parse(PROJECT_DATA_COMPARISON_CONFIG);
        Document result =
mongoCollection.aggregate(Arrays.asList(matchClient, unwind,
matchIndex, project)).first();
        return getFieldFromResult(result,
COMPARISON_CONFIG_FIELD_NAME);
    }

    private String getFieldFromResult(Document resultSet, String name)
{
        if (resultSet == null) {
            return null;
        }

        Document data = (Document) resultSet.get(DATA_FIELD_NAME);
        if (data == null) {
            return null;
        }

        Document value = (Document) data.getOrDefault(name, null);

        if (value != null) {
            LOG.debug("Data was found");
        }

        return value == null ? null : value.toJson();
    }

}

import com.epam.dlex.domain.RequestMetaData;
import com.epam.dlex.domain.SequenceWrapper;
import com.epam.dlex.repository.RequestRepository;
import com.epam.dlex.repository.ResponseRepository;
import com.epam.dlex.repository.imp.ZipBuilder;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.FindOneAndUpdateOptions;
import com.mongodb.client.model.UpdateOptions;
import com.mongodb.client.result.UpdateResult;
import org.bson.Document;
import org.bson.conversions.Bson;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.util.*;

import static com.epam.dlex.repository.imp.mongo.MongoConstants.*;

/**
 * This {@code MongoRequestRepositoryImpl} represents the
implementation of
 * {@link ResponseRepository} interface to storage
 * data in MongoDB.
 */
public class MongoRequestRepositoryImpl implements RequestRepository {

    private MongoCollection<Document> clientsCollection;
```

```
    private static final Logger LOG =
LoggerFactory.getLogger(MongoRequestRepositoryImpl.class);

    private static final String REQUEST_FILE = "request.txt";
    private static final String RESPONSE_FILE = "response.txt";


    public MongoRequestRepositoryImpl(MongoCollection<Document>
clientsCollection) {
        this.clientsCollection = clientsCollection;
    }

    @Override
    public String save(String clientName, String serializedRequest,
RequestMetaData requestMetaData) {
        LOG.debug("Saving request for client {}", clientName);
        UpdateOptions options = new UpdateOptions().upsert(true);

        Bson clientEq = Filters.eq(CLIENT_FIELD_NAME, clientName);

        Document defaultDoc =
Document.parse(String.format(SET_ON_INSERT_REQUEST,
serializedRequest));
        clientsCollection.updateOne(clientEq, defaultDoc, options);

        FindOneAndUpdateOptions findStepOptions = new
FindOneAndUpdateOptions();

findStepOptions.projection(Document.parse(PROJECT_FIND_STEP_REQUEST));

        Document result = clientsCollection.findOneAndUpdate(clientEq,
Document.parse(INC_STEP_REQUEST), findStepOptions);
        Integer index = (Integer) result.get(STEP_FIELD_NAME);
        LOG.debug("Current request step [{}] for client {} ", index,
clientName);

        Document newDocument =
Document.parse(String.format(SAVE_REQUEST, index, serializedRequest));
        clientsCollection.updateOne(clientEq, newDocument, options);

        return index.toString();
    }

    @Override
    public boolean isRootExists(String clientName) {
        Bson filter = Filters.eq(CLIENT_FIELD_NAME, clientName);
        return clientsCollection.find(filter)
                .iterator().hasNext();
    }

    @Override
    public boolean removeRoot(String clientName) {
        Bson filter = Filters.eq(CLIENT_FIELD_NAME, clientName);
        return clientsCollection.deleteOne(filter)
                .getDeletedCount() > 0;
    }

    @Override
    public boolean clearRoot(String clientName) {
        boolean result = deleteData(clientName);
        LOG.trace("Delete 'data' field for client: {}", clientName);
        result &= resetStep(clientName);
        LOG.trace("Delete 'step' field for client: {}", clientName);
        return result;
```

```
    }

    private boolean deleteData(String clientName) {
        Document newDocumentState = Document.parse(DELETE_DATA);
        Bson filter = Filters.eq(CLIENT_FIELD_NAME, clientName);
        UpdateResult result = clientsCollection.updateOne(filter,
newDocumentState);
        return result.getModifiedCount() > 0;
    }

    private boolean resetStep(String clientName) {
        Document newDocumentState = Document.parse(RESET_STEP);
        Bson filter = Filters.eq(CLIENT_FIELD_NAME, clientName);
        UpdateResult result = clientsCollection.updateOne(filter,
newDocumentState);
        return result.getModifiedCount() > 0;
    }


    @Override
    public String read(String clientName, String currentStep) {
        LOG.info("Reading request for client {} with step {}",
clientName, currentStep);
        int step = Integer.parseInt(currentStep);

        Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, clientName));
        Document matchIndex =
Document.parse(String.format(MATCH_INDEX_REQUEST, step));
        Document unwind = Document.parse(UNWIND_REQUEST);
        Document project = Document.parse(PROJECT_REQUEST_REQUEST);

        Document result =
clientsCollection.aggregate(Arrays.asList(matchClient, unwind,
matchIndex, project)).first();

        if (result == null) {
            LOG.info("Request wasn't found for client {} with step {}
in db", clientName, currentStep);
            return null;
        }
        Document data = (Document)
result.getOrDefault(DATA_FIELD_NAME, EMPTY_DOCUMENT);
        Document request = (Document) data.get(REQUEST_FIELD_NAME);

        return request.toJson();
    }

    @Override
    public boolean update(String clientName, String currentStep,
String serializedRequest, RequestMetaData requestMetaData) {
        LOG.debug("Updating request for client {} with step {}",
clientName, currentStep);
        int step = Integer.parseInt(currentStep);

        Bson stepEq = Filters.eq(INDEX_FIELD_NAME, step);
        Bson dataElemMatch = Filters.elemMatch(DATA_FIELD_NAME,
stepEq);
        Bson clientEq = Filters.eq(CLIENT_FIELD_NAME, clientName);
        Bson filter = Filters.and(clientEq, dataElemMatch);

        Document newDocument =
Document.parse(String.format(UPDATE_REQUEST, serializedRequest));
```

```
        UpdateResult result = clientsCollection.updateOne(filter,
newDocument);
        return result.getModifiedCount() > 0;
    }

    @Override
    public InputStream getAllAsZip(String clientName) {
        try {
            LOG.debug("Starting to complete ZIP for " + clientName);
            ZipBuilder builder = new ZipBuilder(clientName);
            Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, clientName));
            Document unwind = Document.parse(UNWIND_REQUEST);
            Document project =
Document.parse(PROJECT_RES_REQ_REQUEST_INDEX);
            int rangeIndex[] = {-DELTA_SHIFT + 1, 1};
            do {
                rangeIndex[0] = rangeIndex[1];
                rangeIndex[1] += DELTA_SHIFT;
                Document matchIndex =
Document.parse(String.format(MATCH_RANGE_INDEX_REQUEST, rangeIndex[0],
rangeIndex[1]));
                AggregateIterable<Document> result =
clientsCollection.aggregate(Arrays.asList(matchClient, unwind,
matchIndex, project));
                for (Document item : result) {
                    Document data = (Document)
item.get(DATA_FIELD_NAME);
                    int index = data.getInteger(INDEX_FIELD_NAME);
                    String request = getFieldFromResult(item,
REQUEST_FIELD_NAME);
                    String response = getFieldFromResult(item,
RESPONSE_FIELD_NAME);
                    builder.addEntry(index + File.separator +
REQUEST_FILE, request);
                    builder.addEntry(index + File.separator +
RESPONSE_FILE, response);
                    rangeIndex[0]++;
                }
            } while (rangeIndex[0] == rangeIndex[1]);

            LOG.debug("Index {}", rangeIndex);
            LOG.debug("ZIP for {} is ready", clientName);
            return builder.build();
        } catch (IOException e) {
            throw new IllegalStateException("Couldn't create temporary
zip file", e);
        }
    }

    @Override
    public List<SequenceWrapper> getSequenceForClient(String
clientName) {
        Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, clientName));
        Document unwind = Document.parse(UNWIND_REQUEST);
        Document project = Document.parse(PROJECT_SEQUENCE);
        AggregateIterable<Document> result =
clientsCollection.aggregate(Arrays.asList(matchClient, unwind,
project));
        List<SequenceWrapper> sequences = new ArrayList<>();
        for(Document document : result){
            SequenceWrapper sequenceWrapper = new SequenceWrapper();
            Document data = document.get("data", Document.class);
```

```
            sequenceWrapper.setStep(data.getInteger("index"));

sequenceWrapper.setMethod(data.get("request",Document.class).getString
("method"));

sequenceWrapper.setUrl(data.get("request",Document.class).getString("p
ath"));
            sequences.add(sequenceWrapper);
        }
        return sequences;
    }

    @Override
    public Set<Integer> getStepSequence(String clientName) {
        Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, clientName));
        Document project = Document.parse(PROJECT_SIZE_REQUEST);
        Document result =
clientsCollection.aggregate(Arrays.asList(matchClient,
project)).first();

        if (result == null) {
            LOG.debug("Sequence was empty for client {}", clientName);
            return Collections.emptySet();
        }

        Integer size = (Integer) result.get(SEQUENCE_SIZE_FIELD);
        Set<Integer> sequence = new HashSet<>(size);
        for (int i = 1; i <= size; i++) {
            sequence.add(i);
        }

        return sequence;
    }

    private String getFieldFromResult(Document resultSet, String name)
{
        if (resultSet == null) {
            return null;
        }

        Document data = (Document)
resultSet.getOrDefault(DATA_FIELD_NAME, EMPTY_DOCUMENT);
        Document value = (Document) data.get(name);

        if (value != null) {
            LOG.debug("Data was found");
        }

        return value == null ? null : value.toJson();
    }
}

import com.epam.dlex.filter.RequestFilter;
import com.epam.dlex.repository.ResponseRepository;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.model.Filters;
import com.mongodb.client.result.UpdateResult;
import org.bson.Document;
import org.bson.conversions.Bson;

import java.util.Arrays;
```

```
import static com.epam.dlex.repository.imp.mongo.MongoConstants.*;

/**
 * Repository implementation for MongoDB.
 *
 * @author Bohdan_Suprun
 */
public class MongoResponseRepositoryImpl implements ResponseRepository
{

    private static final Logger LOG =
LoggerFactory.getLogger(MongoResponseRepositoryImpl.class);

    private final MongoCollection<Document> mongoCollection;

    public MongoResponseRepositoryImpl(MongoCollection<Document>
mongoCollection) {
        this.mongoCollection = mongoCollection;
    }

    @Override
    public boolean save(String requestId, String clientName, String
response) {
        LOG.debug("Saving response for client {} and sequence {}",
clientName, requestId);
        int requestNumber = Integer.parseInt(requestId);

        Document newDocumentState =
Document.parse(String.format(SAVE_RESPONSE, response));
        Bson filter = Filters.and(Filters.eq(CLIENT_FIELD_NAME,
clientName), Filters.elemMatch(DATA_FIELD_NAME,
                Document.parse(String.format(SAVE_REQUEST_CONDITION,
requestNumber))));

        UpdateResult result = mongoCollection.updateOne(filter,
newDocumentState);
        LOG.debug("Rows affected {}", result.getModifiedCount());
        return result.getModifiedCount() > 0;
    }

    @Override
    public String findResponseByRequest(RequestFilter filter, String
client) {
        LOG.debug("Searching response for client {}", client);
        Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, client));
        Document unwind = Document.parse(UNWIND_REQUEST);
        Document project = Document.parse(PROJECT_REQUEST_REQUEST);

        // just for avoiding condition in the loop
        int rangeIndex[] = {- DELTA_SHIFT + 1, 1};

        do {
            rangeIndex[0] = rangeIndex[1];
            rangeIndex[1] += DELTA_SHIFT;

            // Set pagination
            Document matchIndexRange =
Document.parse(String.format(MATCH_RANGE_INDEX_REQUEST, rangeIndex[0],
rangeIndex[1]));
            AggregateIterable<Document> result =
mongoCollection.aggregate(Arrays.asList(matchClient, unwind,
matchIndexRange, project));
```

```
            for (Document item : result) {
                String request = getFieldFromResult(item,
REQUEST_FIELD_NAME);

                if (request == null) {
                    return null;
                }

                if (filter.accept(request)) {
                    LOG.debug("Match found in index {}",
rangeIndex[0]);
                    return read(String.valueOf(rangeIndex[0]),
client);
                }

                rangeIndex[0]++;
            }
            // there are records for retrieving
        } while (rangeIndex[0] == rangeIndex[1]);

        LOG.debug("Nothing was found for client {}", client);
        return null;
    }

    @Override
    public String findResponseByRequestInFolder(RequestFilter filter,
String clientName, String folderName) {
        LOG.debug("Searching response for client {} and sequence {}",
clientName, folderName);
        int folderNumber = Integer.parseInt(folderName);

        Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, clientName));
        Document matchIndex =
Document.parse(String.format(MATCH_INDEX_REQUEST, folderNumber));
        Document unwind = Document.parse(UNWIND_REQUEST);
        Document project = Document.parse(PROJECT_RES_REQ_REQUEST);

        Document result =
mongoCollection.aggregate(Arrays.asList(matchClient, unwind,
matchIndex, project)).first();

        String request = getFieldFromResult(result,
REQUEST_FIELD_NAME);

        if (request != null && filter.accept(request)) {
            LOG.debug("Match found for client {}", clientName);
            return getFieldFromResult(result, RESPONSE_FIELD_NAME);
        }

        LOG.debug("Nothing was found for client {}", clientName);
        return null;
    }

    @Override
    public String read(String requestId, String client) {
        LOG.debug("Reading response for client {} with sequence {}",
client, requestId);
        int folderNumber = Integer.parseInt(requestId);

        Document matchClient =
Document.parse(String.format(MATCH_CLIENT_REQUEST, client));
        Document matchIndex =
Document.parse(String.format(MATCH_INDEX_REQUEST, folderNumber));
```

```
        Document unwind = Document.parse(UNWIND_REQUEST);
        Document project = Document.parse(PROJECT_RESPONSE_REQUEST);

        Document result =
mongoCollection.aggregate(Arrays.asList(matchClient, unwind,
matchIndex, project)).first();
        return getFieldFromResult(result, RESPONSE_FIELD_NAME);
    }

    @Override
    public boolean update(String requestId, String clientName, String
response) {
        return save(requestId, clientName, response);
    }

    private String getFieldFromResult(Document resultSet, String name)
{
        if (resultSet == null) {
            LOG.debug("Result set is empty");
            return null;
        }

        Document data = (Document) resultSet.get(DATA_FIELD_NAME);
        if (data == null) {
            return null;
        }

        Document value = (Document) data.getOrDefault(name, null);

        if (value != null) {
            LOG.debug("Data was found");
        }

        return value == null ? null : value.toJson();
    }
}

public interface MongoConstants {

    String CLIENT_FIELD_NAME = "client";

    String DATA_FIELD_NAME = "data";

    String RESPONSE_FIELD_NAME = "response";

    String REQUEST_FIELD_NAME = "request";

    String COMPARISON_CONFIG_FIELD_NAME = "rule";

    String SEQUENCE_SIZE_FIELD = "sequenceSize";

    String SAVE_RESPONSE = "{'$set':{'data.$.response':%s}}";

    String INDEX_FIELD_NAME = "index";

    String STEP_FIELD_NAME = "step";

    String SAVE_REQUEST =
"{'$push':{'data':{'index':%d,'request':%s}}}";

    String SET_ON_INSERT_REQUEST = "{'$setOnInsert':{'data':[],
'step':1}}";

    String INC_STEP_REQUEST = "{'$inc':{'step':1}}";
```

```
    String PROJECT_RES_REQ_REQUEST_INDEX =
"{'$project':{'data.index':1, 'data.response':1, 'data.request':1}}";

    String PROJECT_FIND_STEP_REQUEST = "{'step':1,'_id':0}";

    String UNWIND_REQUEST = "{'$unwind':{'path':'$data'}}";

    String PROJECT_SEQUENCE = "{'$project':{'data.index':1,
'data.request.path':1,'data.request.method':1, '_id':0}}";

    String SAVE_REQUEST_CONDITION = "{'index': %d}";

    String MATCH_CLIENT_REQUEST = "{'$match':{'client':'%s'}}";

    String MATCH_INDEX_REQUEST = "{'$match':{'data.index':%d}}";

    String UPDATE_REQUEST = "{'$set':{'data.$.request':%s}}";

    String PROJECT_SIZE_REQUEST =
"{'$project':{'sequenceSize':{'$size':'$data'}}}";

    Document EMPTY_DOCUMENT = new Document();

    String MATCH_RANGE_INDEX_REQUEST =
"{'$match':{'$and':[{'data.index':{'$gte':%d}},{'data.index':{'$lt':%d
}}]}}";

    String PROJECT_RESPONSE_REQUEST = "{'$project':{'data.response':1,
'_id':0}}";

    String PROJECT_REQUEST_REQUEST = "{'$project':{'data.request':1,
'_id':0}}";

    String PROJECT_RES_REQ_REQUEST = "{'$project':{'data.response':1,
'data.request':1, '_id':0}}";

    String SAVE_COMPARISON_CONFIG = "{'$set':{'rule':%s}}";

    String SAVE_COMPARISON_CONFIG_BY_STEP =
"{'$set':{'data.$.rule':%s}}";

    String SAVE_COMPARISON_CONFIG_BY_STEP_CONDITION = "{'index': %d}";

    String DELETE_COMPARISON_CONFIG = "{'$unset':{'rule':1}}";

    String DELETE_RULE_SEQUENCE = "{'$unset':{'data.rule':1}}";

    String DELETE_COMPARISON_CONFIG_BY_STEP =
"{'$unset':{'data.$.rule':1}}";

    String RESET_STEP = "{'$set':{'step':1}}";

    String DELETE_DATA = "{'$unset':{'data':1}}";

    String PROJECT_COMPARISON_CONFIG = "{'$project':{'rule':1,
'_id':0}}";

    String PROJECT_DATA_COMPARISON_CONFIG =
"{'$project':{'data.rule':1, '_id':0}}";

    int DELTA_SHIFT = 5;
}
```

Додаток В

Відомість магістрської роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	83с.
		.
	Інші документи	
2.	Презентаційні матеріали	16 плакатів

					Дослідження ассортативності безмаштабних мереж		
.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Черномуров Д.А.				Шифр групи	Код напр./спец.
Перевірив		Удовенко С.Г.				СШІмзд-18-1	122
Н.контр.		.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					