**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное учреждение высшего образования

**“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИТМО**”

**Факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА № 6**

**ПО ДИСЦИПЛИНЕ «ТЕХНОЛОГИИ ВЕБ-СЕРВИСОВ»**

Студент: Су Минцзе

Группа: P41141

Преподаватель: Дергачев Андрей Михайлович

Санкт-Петербург

2021

**Задание:**

Необходимо выполнить задание из третьей работы, но с использованием

REST-сервиса. Таблицу базы данных, а также код для работы с ней можно оставить без изменений.

**Выполнение работы**:

В результате выполнения пятой лабораторной работы был написан REST-сервис в виде standalone-приложения с операциями CRUD. Класс App.java содержит метод main, осуществляет запуск сервиса и регистрирует обработчик ошибок. В классы PictureWebServise.java и PostgreSQLDAO.java были добавлены исключительные ситуации:

• метод getOne, в случае если не установлен id, выдает исключение IllegalIdException. В случае успеха возвращает данные с указанным id.

• Метод create, в случае если не установлен один из параметров для создания записи, выдает исключение InvalidCreatingParametersException и указывает на параметр. Если привставке такой id уже есть в таблице, то выдает исключение InsertingException. В случае успеха возвращает id новой записи.

• Метод update в случае, если не установлен id или полностью данные для обновления записи, выдает исключения IllegalIdException и InvalidUpdatingParametersException соответственно. Если при обновлении записи не найден указанный id, то выдает исключение InvalidEntityException. В случае успеха возвращает количество обновленных записей.

• Метод delete в случае, если не установлен id, выдает исключение IllegalIdException. Если при удалении записи не найден указанный id, то выдает исключение InvalidEntityException. В случае успеха возвращает количество удаленных записей.Классы IllegalIdException.java, InvalidUpdatingParametersException.java, InvalidCreatingParametersException.java, InsertingException.java и InvalidEntityException.java – собственные классы исключения, наследники java.lang.Exception. Классы IllegalIdMapper.java,InvalidUpdatingParametersMapper.java,InvalidCreatingParametersMapper.java, InsertingMapper.java и InvalidEntityMapper.java – классы реализаций интерфейса ExceptionMapper, которые представляют собой универсальные обработчики ошибок,описанные выше. Класс BasicResponse.java представляет из себя ответ во время

исключительной ситуации в JSON-формате. Содержит внутренний статус и сообщение.

Классы Picture.java и ConnectionUtil.java не изменились. Исходный код

классов REST-сервиса представлен в листингах 6.1-6.14.

Листинг 6.1 – Файл App.java

package com.sumingjie.service;

import com.sun.jersey.api.container.grizzly2.GrizzlyServerFactory;

import com.sun.jersey.api.core.PackagesResourceConfig;

import com.sun.jersey.api.core.ResourceConfig;

import java.io.IOException;

import java.net.URI;

import org.glassfish.grizzly.http.server.HttpServer;

public class App

{

private static final URI BASE\_URI = URI.create("http://localhost:8080/");

public static void main(String[] args) {

HttpServer server = null;

try {

ResourceConfig resourceConfig = new

PackagesResourceConfig(PictureResource.class.getPackage().getName());

server = GrizzlyServerFactory.createHttpServer(BASE\_URI, resourceConfig);

server.start();

System.in.read();

stopServer(server);

} catch (IOException e) {

e.printStackTrace();

stopServer(server);

}

}

private static void stopServer(HttpServer server) {

if (server != null)

server.stop(); }}

Листинг 6.2 – Файл BasicResponse.java

package com.sumingjie.service;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="BasicResponce")

public class BasicResponse {

public String internalStatus;

public String message;

public BasicResponse() {}

public BasicResponse(String message){

this.internalStatus = "Error";

this.message = message;

}

}

Листинг 6.3 – Файл PictureResource.java

package com.sumingjie.service;

import com.sumingjie.service.exceptions.AuthException;

import com.sumingjie.service.exceptions.IllegalIdException;

import com.sumingjie.service.exceptions.InsertingException;

import com.sumingjie.service.exceptions.InvalidCreatingParametersException;

import com.sumingjie.service.exceptions.InvalidEntityException;

import com.sumingjie.service.exceptions.InvalidUpdatingParametersException;

import java.util.List;

import java.util.StringTokenizer;

import javax.ws.rs.Consumes;

import javax.ws.rs.DELETE;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.PUT;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.HttpHeaders;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.UriInfo;

import org.postgresql.util.Base64;

@Path("/pictures")

@Produces({MediaType.APPLICATION\_JSON})

public class PictureResource {

private static final String AUTH\_HEADER\_KEY = "Authorization";

private static final String AUTH\_HEADER\_PREFIX = "Basic ";

private static final String USERNAME = "admin";

private static final String PASSWORD = "admin";

@GET

public List<Picture> find(@Context UriInfo info) {

String id = info.getQueryParameters().getFirst("id");

String name = info.getQueryParameters().getFirst("name");

String author = info.getQueryParameters().getFirst("author");

String year = info.getQueryParameters().getFirst("year");

String material = info.getQueryParameters().getFirst("material");

String height = info.getQueryParameters().getFirst("height");

String width = info.getQueryParameters().getFirst("width");

return new PostgreSQLDAO().findPictures(id, name, author, year, material, height, width);

}

@GET

@Path("/{id}")

public List<Picture> getOne(@PathParam("id") int id) throws IllegalIdException {

if (id <= 0) {

throw IllegalIdException.DEFAULT\_INSTANCE;

}

return new PostgreSQLDAO().findOne(id);

}

@POST

@Consumes("application/json")

public String create(@Context HttpHeaders headers, Picture picture) throws InvalidCreatingParametersException,

InsertingException, AuthException {

if (!isAuthenticated(headers)) {

throw AuthException.DEFAULT\_INSTANCE;

}

if (picture.getName() == null || picture.getName().trim().isEmpty()) {

throw new InvalidCreatingParametersException("Invalid creating parameter: name");

}

if (picture.getAuthor() == null || picture.getAuthor().trim().isEmpty()) {

throw new InvalidCreatingParametersException("Invalid creating parameter: author");

}

if (picture.getYear() <= 0) {

throw new InvalidCreatingParametersException("Invalid creating parameter: year");

}

if (picture.getMaterial() == null || picture.getMaterial().trim().isEmpty()) {

throw new InvalidCreatingParametersException("Invalid creating parameter: material");

}

if (picture.getHeight() <= 0) {

throw new InvalidCreatingParametersException("Invalid creating parameter: height");

}

if (picture.getWidth() <= 0) {

throw new InvalidCreatingParametersException("Invalid creating parameter: width");

}

PostgreSQLDAO dao = new PostgreSQLDAO();

return "{\"result\":" + dao.createPicture(picture) + "}";

}

@PUT

@Consumes("application/json")

@Path("/{id}")

public String update(@Context HttpHeaders headers, @PathParam("id") int id, Picture picture) throws IllegalIdException, InvalidEntityException,

InvalidUpdatingParametersException, AuthException {

if (!isAuthenticated(headers)) {

throw AuthException.DEFAULT\_INSTANCE;

}

if (id <= 0) {

throw IllegalIdException.DEFAULT\_INSTANCE;

}

if ((picture.getName() == null || picture.getName().trim().isEmpty()) &&

(picture.getAuthor() == null || picture.getAuthor().trim().isEmpty()) &&

(picture.getYear() <= 0) &&

(picture.getMaterial() == null || picture.getMaterial().trim().isEmpty()) &&

(picture.getHeight() <= 0) &&

(picture.getWidth() <= 0)) {

throw InvalidUpdatingParametersException.DEFAULT\_INSTANCE;

}

PostgreSQLDAO dao = new PostgreSQLDAO();

return "{\"result\":" + dao.updatePicture(id, picture) + "}";

}

@DELETE

@Path("/{id}")

public String delete(@Context HttpHeaders headers, @PathParam("id") int id) throws IllegalIdException,

InvalidEntityException, AuthException {

if (!isAuthenticated(headers)) {

throw AuthException.DEFAULT\_INSTANCE;

}

if (id <= 0) {

throw IllegalIdException.DEFAULT\_INSTANCE;

}

PostgreSQLDAO dao = new PostgreSQLDAO();

return "{\"result\":" + dao.deletePicture(id) + "}";

}

private boolean isAuthenticated(HttpHeaders headers) {

List<String> authHeader = headers.getRequestHeaders().get(AUTH\_HEADER\_KEY);

if (authHeader == null) {

return false;

}

String authToken = authHeader.get(0);

if (authToken.isEmpty()) {

return false;

}

authToken = authToken.replaceFirst(AUTH\_HEADER\_PREFIX, "");

String decodedString = new String(Base64.decode(authToken));

StringTokenizer stringTokenizer = new StringTokenizer(decodedString, ":");

String username = stringTokenizer.nextToken();

String password = stringTokenizer.nextToken();

return username.equals(USERNAME) && password.equals(PASSWORD); }}

Листинг 6.4 – Файл PostgreSQLDAO.java

package com.sumingjie.service;

import com.sumingjie.service.exceptions.InsertingException;

import com.sumingjie.service.exceptions.InvalidEntityException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

public class PostgreSQLDAO {

private Connection connection;

PostgreSQLDAO() {

this.connection = ConnectionUtil.getConnection();

}

public List<Picture> findPictures(String id, String name, String author, String year, String material, String height, String width) {

StringBuilder sb = new StringBuilder("");

StringBuilder query = new StringBuilder("");

boolean where = false;

if (id != null) {

sb.append("id = ").append(Integer.parseInt(id)).append(" AND ");

where = true;

}

if (name != null) {

sb.append("name = '").append(name).append("' AND ");

where = true;

}

if (author != null) {

sb.append("author = '").append(author).append("' AND ");

where = true;

}

if (year != null) {

sb.append("year = ").append(Integer.parseInt(year)).append(" AND ");

where = true;

}

if (material != null) {

sb.append("material = '").append(material).append("' AND ");

where = true;

}

if (height != null) {

sb.append("height = ").append(Float.parseFloat(height)).append(" AND ");

where = true;

}

if (width != null) {

sb.append("width = ").append(Float.parseFloat(width)).append(" AND ");

where = true;

}

if (where) {

if (sb.toString().endsWith(" AND ")) {

sb.setLength(sb.length() - 5);

}

query.append("SELECT \* FROM pictures WHERE ").append(sb.toString());

} else {

query.append("SELECT \* FROM pictures");

}

return executeQuery(query.toString());

}

public List<Picture> findOne(int id) {

String query = "SELECT \* FROM pictures WHERE id = " + id;

return executeQuery(query);

}

public int createPicture(Picture picture) throws InsertingException {

String sql = "INSERT INTO pictures (name, author, year, material, height, width) VALUES(?, ?, ?, ?, ?, ?)";

PreparedStatement preparedStatement = null;

int id = 0;

try {

preparedStatement = this.connection.prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, picture.getName());

preparedStatement.setString(2, picture.getAuthor());

preparedStatement.setInt(3, picture.getYear());

preparedStatement.setString(4, picture.getMaterial());

preparedStatement.setFloat(5, picture.getHeight());

preparedStatement.setFloat(6, picture.getWidth());

int affectedRows = preparedStatement.executeUpdate();

ResultSet generatedKeys = preparedStatement.getGeneratedKeys();

if (generatedKeys.next()) {

id = (int) generatedKeys.getLong(1);

}

} catch (SQLException e) {

e.printStackTrace();

}

if (id == 0) {

throw InsertingException.DEFAULT\_INSTANCE;

}

return id;

}

public int updatePicture(int id, Picture picture) throws InvalidEntityException {

String sql = "UPDATE pictures SET" + createUpdateQuery(picture) + " WHERE id=?";

PreparedStatement preparedStatement = null;

int affectedRows = 0;

try {

preparedStatement = this.connection.prepareStatement(sql);

preparedStatement.setInt(1, id);

affectedRows = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (affectedRows == 0) {

throw InvalidEntityException.DEFAULT\_INSTANCE;

}

return affectedRows;

}

public int deletePicture(int id) throws InvalidEntityException {

String sql = "DELETE FROM pictures WHERE id = ?";

PreparedStatement preparedStatement = null;

int affectedRows = 0;

try {

preparedStatement = this.connection.prepareStatement(sql);

preparedStatement.setInt(1, id);

affectedRows = preparedStatement.executeUpdate();

} catch (SQLException e) {

e.printStackTrace();

}

if (affectedRows == 0) {

throw InvalidEntityException.DEFAULT\_INSTANCE;

}

return affectedRows;

}

private List<Picture> executeQuery(String sql) {

List<Picture> pictures = new ArrayList<>();

try {

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(sql);

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String author = rs.getString("author");

int year = rs.getInt("year");

String material = rs.getString("material");

float height = rs.getFloat("height");

float width = rs.getFloat("width");

Picture picture = new Picture(id, name, author, year, material, height, width);

pictures.add(picture);

}

} catch (SQLException ex) {

Logger.getLogger(PostgreSQLDAO.class.getName()).log(Level.SEVERE, null, ex);

}

return pictures;

}

private String createUpdateQuery(Picture picture) {

StringBuilder stringBuilderField = new StringBuilder("(");

StringBuilder stringBuilderValues = new StringBuilder("(");

if (picture.getId() > 0) {

stringBuilderField.append("id,");

stringBuilderValues.append(picture.getId()).append(",");

}

if (picture.getName() != null) {

stringBuilderField.append("name,");

stringBuilderValues.append("'").append(picture.getName()).append("',");

}

if (picture.getAuthor() != null) {

stringBuilderField.append("author,");

stringBuilderValues.append("'").append(picture.getAuthor()).append("',");

}

if (picture.getYear() > 0) {

stringBuilderField.append("year,");

stringBuilderValues.append(picture.getYear()).append(",");

}

if (picture.getMaterial() != null) {

stringBuilderField.append("material,");

stringBuilderValues.append("'").append(picture.getMaterial()).append("',");

}

if (picture.getHeight() > 0) {

stringBuilderField.append("height,");

stringBuilderValues.append(picture.getHeight()).append(",");

}

if (picture.getWidth() > 0) {

stringBuilderField.append("width,");

stringBuilderValues.append(picture.getWidth()).append(",");

}

if (stringBuilderField.toString().endsWith(",")) {

stringBuilderField.setLength(stringBuilderField.length() - 1);

stringBuilderValues.setLength(stringBuilderValues.length() - 1);

}

stringBuilderField.append(")");

stringBuilderValues.append(")");

return stringBuilderField.toString() + " = " + stringBuilderValues.toString();}}

Листинг 6.5 – Файл IllegalIdException.java

package com.sumingjie.service.exceptions;

public class IllegalIdException extends Exception {

private static final long serialVersionUID = -6647544772732631047L;

public static IllegalIdException DEFAULT\_INSTANCE = new

IllegalIdException("Parameter id cannot be null or empty");

IllegalIdException(String message) {

super(message); }}

Листинг 6.6 – Файл IllegalIdMapper.java

package com.sumingjie.service.exceptions;

import com.sumingjie.service.BasicResponse;

import com.sun.jersey.api.client.ClientResponse;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.ext.ExceptionMapper;

import javax.ws.rs.ext.Provider;

@Provider

public class IllegalIdMapper implements ExceptionMapper<IllegalIdException> {

@Override

public Response toResponse(IllegalIdException e) {

return Response.status(ClientResponse.Status.BAD\_REQUEST).entity(new BasicResponse(e.getMessage())).type(MediaType.APPLICATION\_JSON).build(); }}

Листинг 6.7 – Файл InvalidUpdatingParametersException.java

package com.sumingjie.service.exceptions;

public class InvalidUpdatingParametersException extends Exception {

private static final long serialVersionUID = -6647544772732631047L;

public static InvalidUpdatingParametersException DEFAULT\_INSTANCE = new

InvalidUpdatingParametersException("Invalid updating parameters");

InvalidUpdatingParametersException(String message) {

super(message); }}

Листинг 6.8 – Файл InvalidUpdatingParametersMapper.java

package com.sumingjie.service.exceptions;

import com.sumingjie.service.BasicResponse;

import com.sun.jersey.api.client.ClientResponse;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.ext.ExceptionMapper;

import javax.ws.rs.ext.Provider;

@Provider

public class InvalidUpdatingParametersMapper implements ExceptionMapper<InvalidUpdatingParametersException> {

@Override

public Response toResponse(InvalidUpdatingParametersException e) {

return Response.status(ClientResponse.Status.BAD\_REQUEST).entity(new BasicResponse(e.getMessage())).type(MediaType.APPLICATION\_JSON).build(); }}

Листинг 6.9 – Файл InvalidCreatingParametersException.java

package com.sumingjie.service.exceptions;

public class InvalidCreatingParametersException extends Exception {

private static final long serialVersionUID = -6647544772732631047L;

public static InvalidCreatingParametersException DEFAULT\_INSTANCE = new

InvalidCreatingParametersException("Invalid creating parameter");

public InvalidCreatingParametersException(String message) {

super(message); }}

Листинг 6.10 – Файл InvalidCreatingParametersMapper.java

package com.sumingjie.service.exceptions;

import com.sumingjie.service.BasicResponse;

import com.sun.jersey.api.client.ClientResponse;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.ext.ExceptionMapper;

import javax.ws.rs.ext.Provider;

@Provider

public class InvalidCreatingParametersMapper implements ExceptionMapper<InvalidCreatingParametersException> {

@Override

public Response toResponse(InvalidCreatingParametersException e) {

return Response.status(ClientResponse.Status.BAD\_REQUEST).entity(new BasicResponse(e.getMessage())).type(MediaType.APPLICATION\_JSON).build(); }}

Листинг 6.11 – Файл InsertingException.java

package com.sumingjie.service.exceptions;

public class InsertingException extends Exception {

private static final long serialVersionUID = -6647544772732631047L;

public static InsertingException DEFAULT\_INSTANCE = new

InsertingException("Error During creation entity");

InsertingException(String message) {

super(message); }}

Листинг 6.12 – Файл InsertingMapper.java

package com.sumingjie.service.exceptions;

import com.sumingjie.service.BasicResponse;

import com.sun.jersey.api.client.ClientResponse;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.ext.ExceptionMapper;

import javax.ws.rs.ext.Provider;

@Provider

public class InsertingMapper implements ExceptionMapper<InsertingException> {

@Override

public Response toResponse(InsertingException e) {

return Response.status(ClientResponse.Status.BAD\_REQUEST).entity(new BasicResponse(e.getMessage())).type(MediaType.APPLICATION\_JSON).build(); } }

Листинг 6.13 – Файл InvalidEntityException.java

package com.sumingjie.service.exceptions;

public class InvalidEntityException extends Exception {

private static final long serialVersionUID = -6647544772732631047L;

public static InvalidEntityException DEFAULT\_INSTANCE = new

InvalidEntityException("Invalid entity");

InvalidEntityException(String message) {

super(message); } }

Листинг 6.14 – Файл InvalidEntityMapper.java

package com.sumingjie.service.exceptions;

import com.sumingjie.service.BasicResponse;

import com.sun.jersey.api.client.ClientResponse;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.ext.ExceptionMapper;

import javax.ws.rs.ext.Provider;

@Provider

public class InvalidEntityMapper implements ExceptionMapper<InvalidEntityException> {

@Override

public Response toResponse(InvalidEntityException e) {

return Response.status(ClientResponse.Status.BAD\_REQUEST).entity(new BasicResponse(e.getMessage())).type(MediaType.APPLICATION\_JSON).build(); } }

Код клиента содержит файлы Picture.java, полученный в результате выполнения предыдущей лабораторной работы. В App.java, исходный код которого представлен в листинге 6.15, был изменен только метод main.

В классе App.java последовательно выполняются запросы:

• метод POST /pictures, данные в JSON-формате: name=Богатыри, author=Виктор Михайлович Васнецов;

• метод POST /pictures, данные в JSON-формате: name=Богатыри, author=Виктор Михайлович Васнецов, year=1881, material=Маслянные краски, height=295.3, width=446;

• метод PUT /pictures/10, данные в JSON-формате: нет;

• метод PUT /pictures/111, данные в JSON-формате: name=My own picture, author=ITMO, year=2018;

• метод PUT /pictures/10, данные в JSON-формате: name=My own picture, author=ITMO, year=2018;

• метод DELETE /pictures/111;

• метод DELETE /pictures/10.

Листинг 6.15 – App.java

package com.sumingjie.client;

import com.sun.jersey.api.client.Client;

import com.sun.jersey.api.client.ClientResponse;

import com.sun.jersey.api.client.GenericType;

import com.sun.jersey.api.client.WebResource;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Set;

import javax.ws.rs.core.MediaType;

import org.postgresql.util.Base64;

public class App {

private static final String URL = "http://localhost:8080/pictures";

private static final String USERNAME = "admin";

private static final String PASSWORD = "admin";

public static void main(String[] args) {

String authToken = "Basic " + Base64.encodeBytes((USERNAME + ":" + PASSWORD).getBytes());

Client client = Client.create();

System.out.println("Simple hard code client for service");

status(client);

System.out.println("Query with auth: /pictures, \nMethod: POST, \nData: name=Богатыри, author=Виктор Михайлович Васнецов");

String json = "{\"name\":\"Богатыри\", \"author\":\"Виктор Михайлович Васнецов\"}";

System.out.println("Result: " + sendRequest(client, URL, authToken, "POST", json));

System.out.println();

status(client);

System.out.println("Query without auth: /pictures, \nMethod: POST, \nData: name=Богатыри, author=Виктор Михайлович Васнецов, " +

"year=1881, material=Маслянные краски, height=295.3, width=446");

json = "{\"name\":\"Богатыри\"," +

"\"author\":\"Виктор Михайлович Васнецов\"," +

"\"year\":1881, " +

"\"material\":\"Маслянные краски\"," +

"\"height\":295.3, " +

"\"width\":446}";

System.out.println("Result: " + sendRequest(client, URL, "", "POST", json));

System.out.println();

status(client);

System.out.println("Query with auth: /pictures, \nMethod: POST, \nData: name=Богатыри, author=Виктор Михайлович Васнецов, " +

"year=1881, material=Маслянные краски, height=295.3, width=446");

System.out.println("Result: " + sendRequest(client, URL, authToken, "POST", json));

System.out.println();

status(client);

System.out.println("Query with auth: /pictures/10, \nMethod: PUT, \nData: none");

System.out.println("Result: " + sendRequest(client, URL + "/10", authToken, "PUT", "{}"));

System.out.println();

status(client);

System.out.println("Query with auth: /pictures/111, \nMethod: PUT, \nData: name=My own picture, author=ITMO, year=2018");

json = "{\"name\":\"My own picture\"," +

"\"author\":\"ITMO\"," +

"\"year\":2018}";

System.out.println("Result: " + sendRequest(client, URL + "/111", authToken, "PUT", json));

System.out.println();

status(client);

System.out.println("Query without auth: /pictures/10, \nMethod: PUT, \nData: name=My own picture, author=ITMO, year=2018");

System.out.println("Result: " + sendRequest(client, URL + "/10", "", "PUT", json));

System.out.println();

status(client);

System.out.println("Query with auth: /pictures/10, \nMethod: PUT, \nData: name=My own picture, author=ITMO, year=2018");

System.out.println("Result: " + sendRequest(client, URL + "/10", authToken, "PUT", json));

System.out.println();

status(client);

System.out.println("Query with auth: /pictures/111, \nMethod: DELETE");

System.out.println("Result: " + sendRequest(client, URL + "/111", authToken, "DELETE", ""));

System.out.println();

status(client);

System.out.println("Query without auth: /pictures/10, \nMethod: DELETE");

System.out.println("Result: " + sendRequest(client, URL + "/10", "", "DELETE", ""));

System.out.println();

status(client);

System.out.println("Query with auth: /pictures/10, \nMethod: DELETE");

System.out.println("Result: " + sendRequest(client, URL + "/10", authToken, "DELETE", ""));

System.out.println();

status(client);

}

private static void status(Client client) {

System.out.println("Query: /pictures, \nMethod: GET");

display(findPictures(client, URL, ""));

System.out.println();

}

private static String sendRequest(Client client, String url, String auth, String method, String json) {

WebResource webResource = client.resource(url);

ClientResponse response = null;

if (method.equals("POST"))

response = webResource.type(MediaType.APPLICATION\_JSON).header("Authorization", auth).post(ClientResponse.class, json);

if (method.equals("PUT"))

response = webResource.type(MediaType.APPLICATION\_JSON).header("Authorization", auth).put(ClientResponse.class, json);

if (method.equals("DELETE"))

response = webResource.type(MediaType.APPLICATION\_JSON).header("Authorization", auth).delete(ClientResponse.class);

if (response != null) {

if ((response.getStatus() != ClientResponse.Status.OK.getStatusCode()) &&

response.getStatus() != ClientResponse.Status.BAD\_REQUEST.getStatusCode()) {

throw new IllegalStateException("Request failed");

}

return response.getEntity(String.class);

}

return "Please specify method type (POST, PUT, DELETE)";

}

private static List<Picture> findPictures(Client client, String url, String query) {

WebResource webResource = client.resource(url);

if (!query.isEmpty()) {

Map<String, String> map = getQueryMap(query);

Set<String> keys = map.keySet();

for (String key : keys) {

webResource = webResource.queryParam(key, map.get(key));

}

}

ClientResponse response = webResource.accept(MediaType.APPLICATION\_JSON).get(ClientResponse.class);

if (response.getStatus() != ClientResponse.Status.OK.getStatusCode()) {

throw new IllegalStateException("Request failed");

}

GenericType<List<Picture>> type = new GenericType<List<Picture>>() {

};

return response.getEntity(type);

}

private static Map<String, String> getQueryMap(String query) {

String[] params = query.split("&");

Map<String, String> map = new HashMap<String, String>();

for (String param : params) {

String name = param.split("=")[0];

String value = param.split("=")[1];

map.put(name, value);

}

return map;

}

private static void display(List<Picture> pictures) {

for (Picture picture : pictures) {

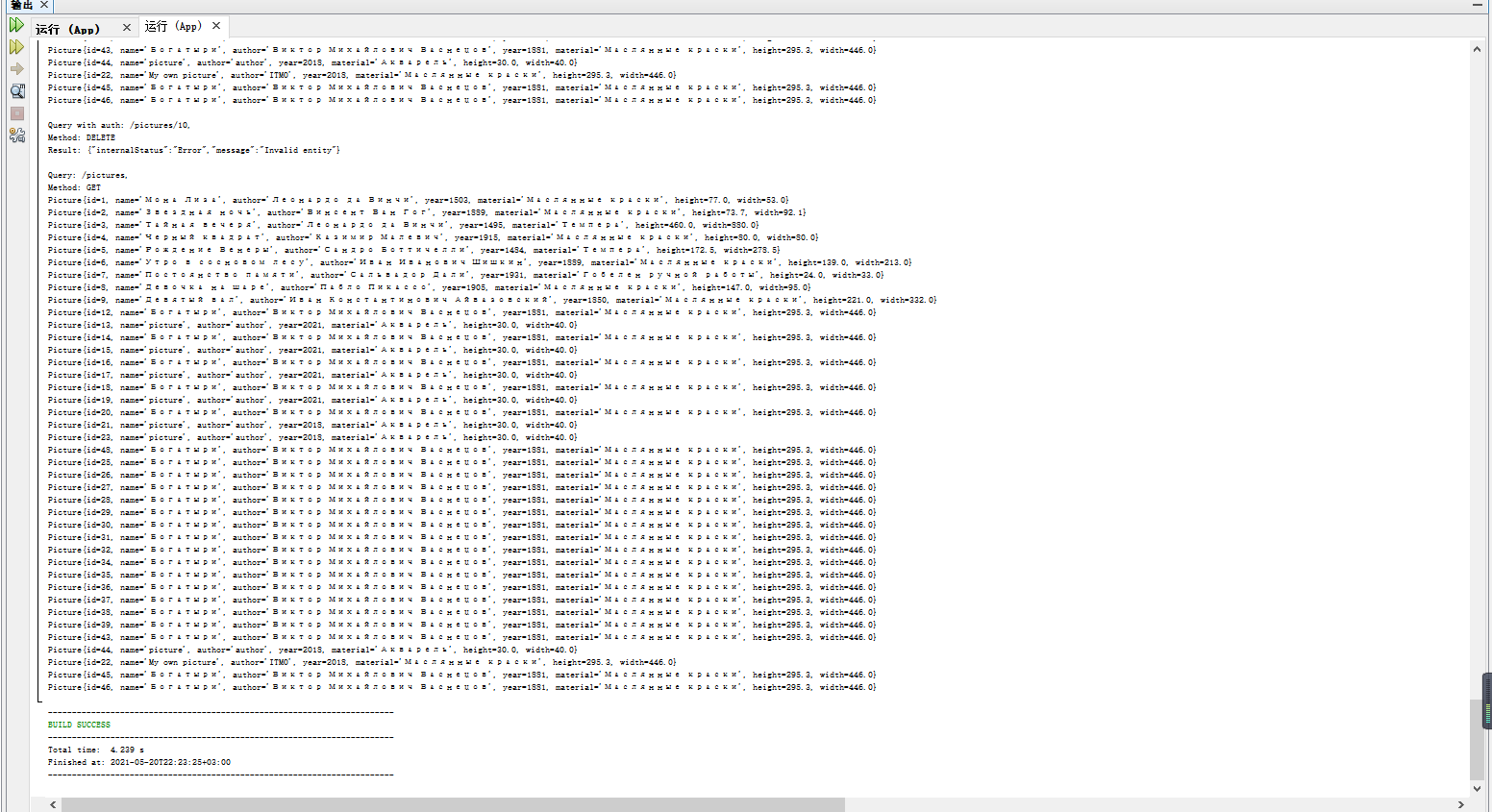
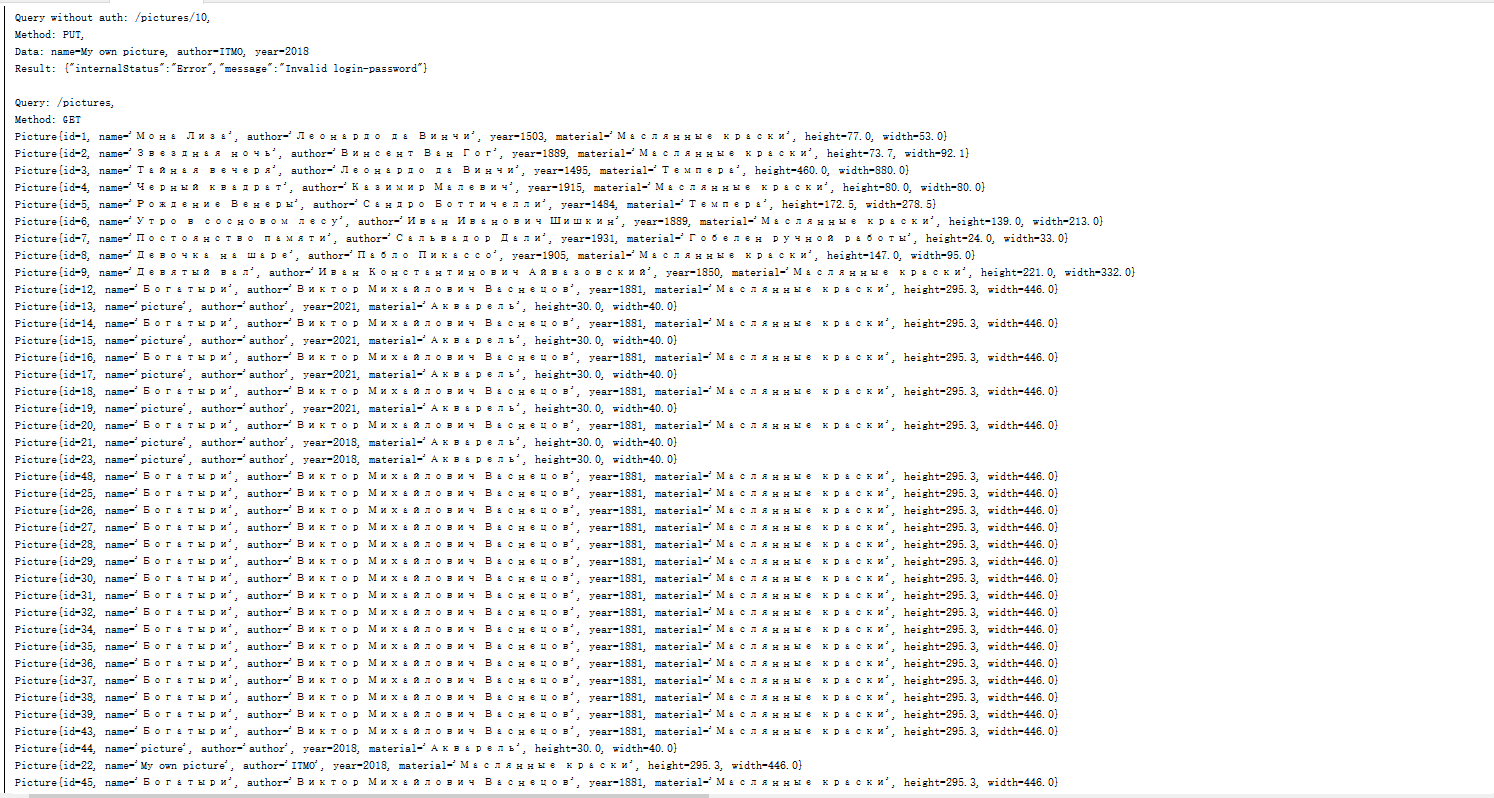
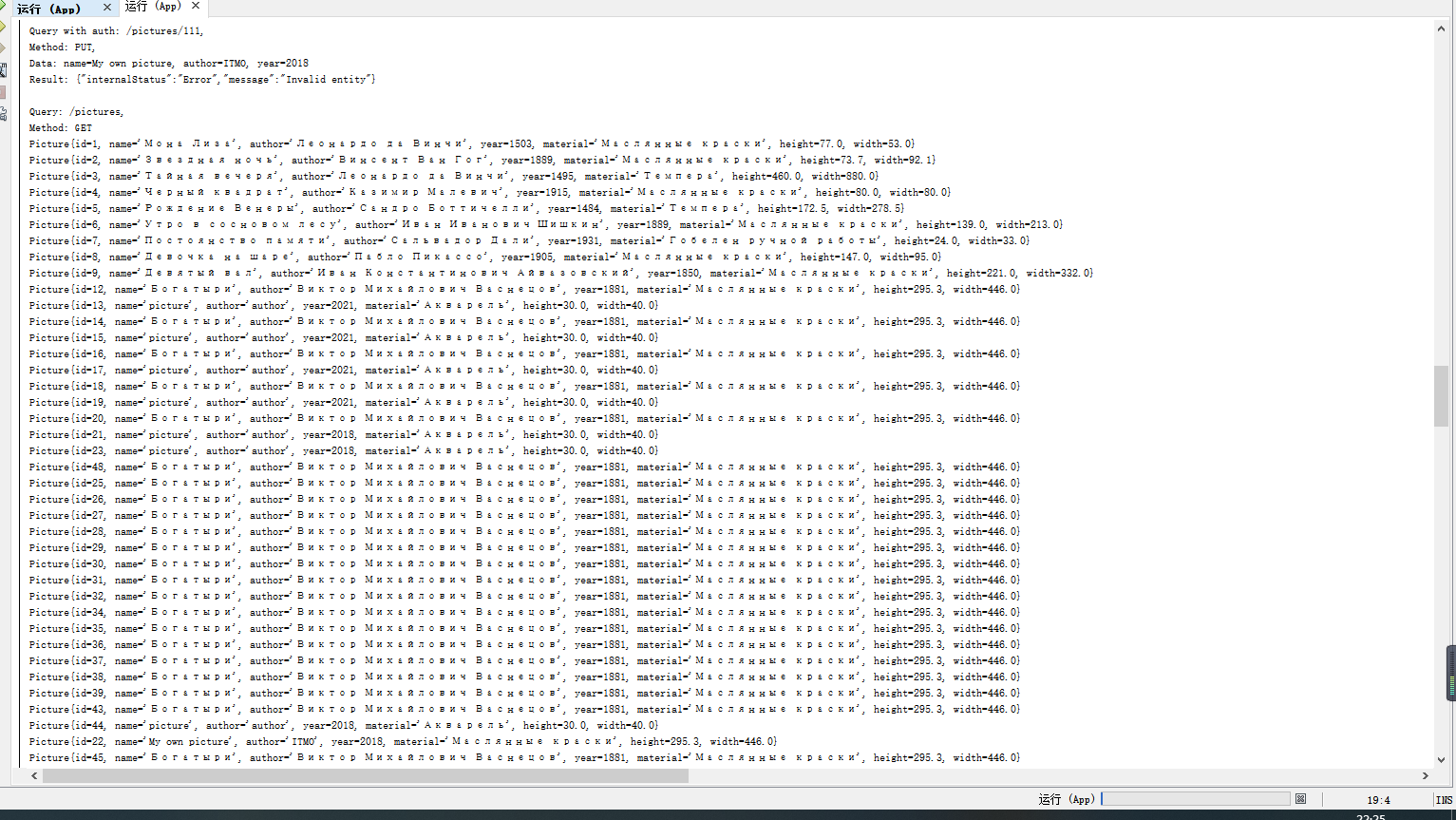
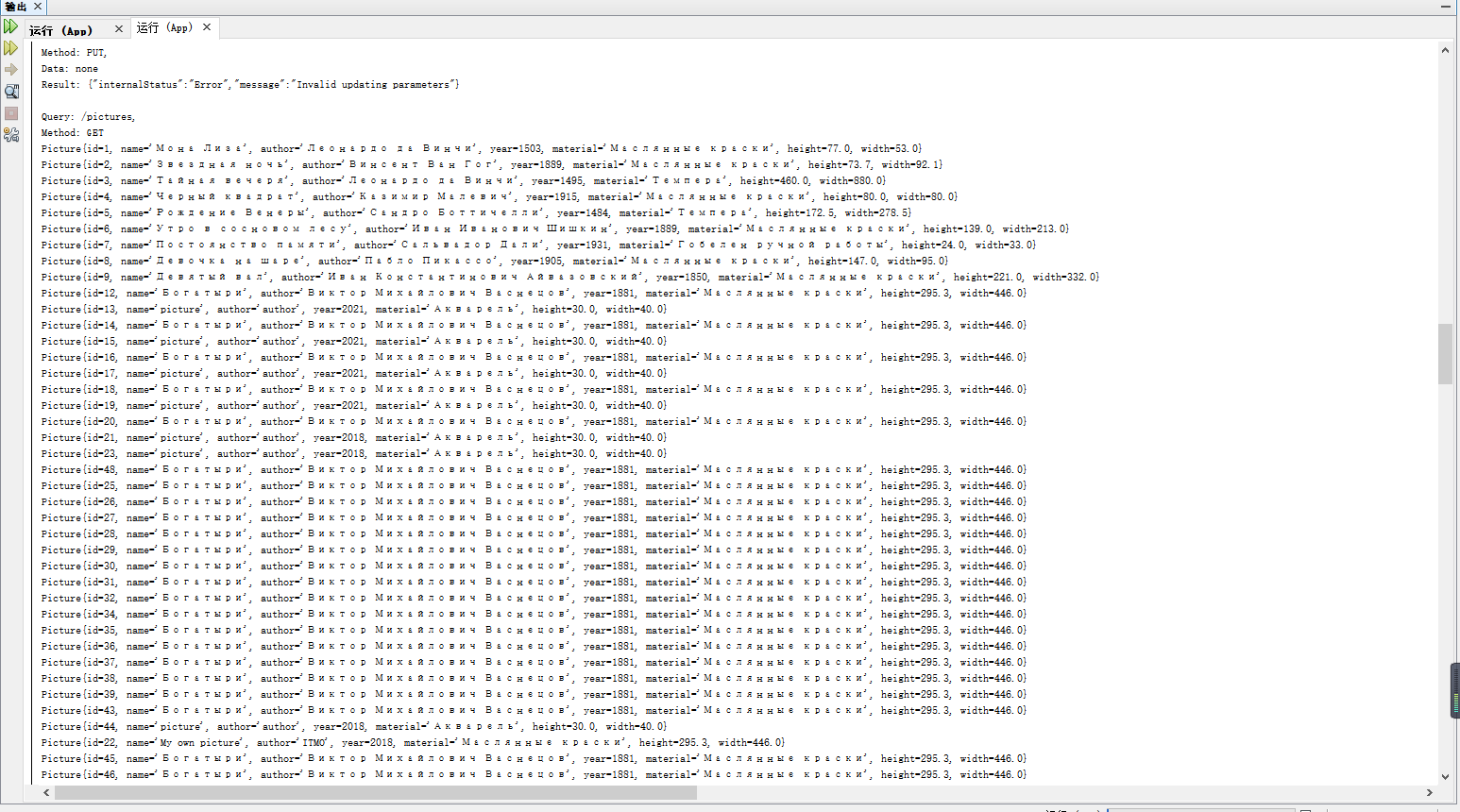
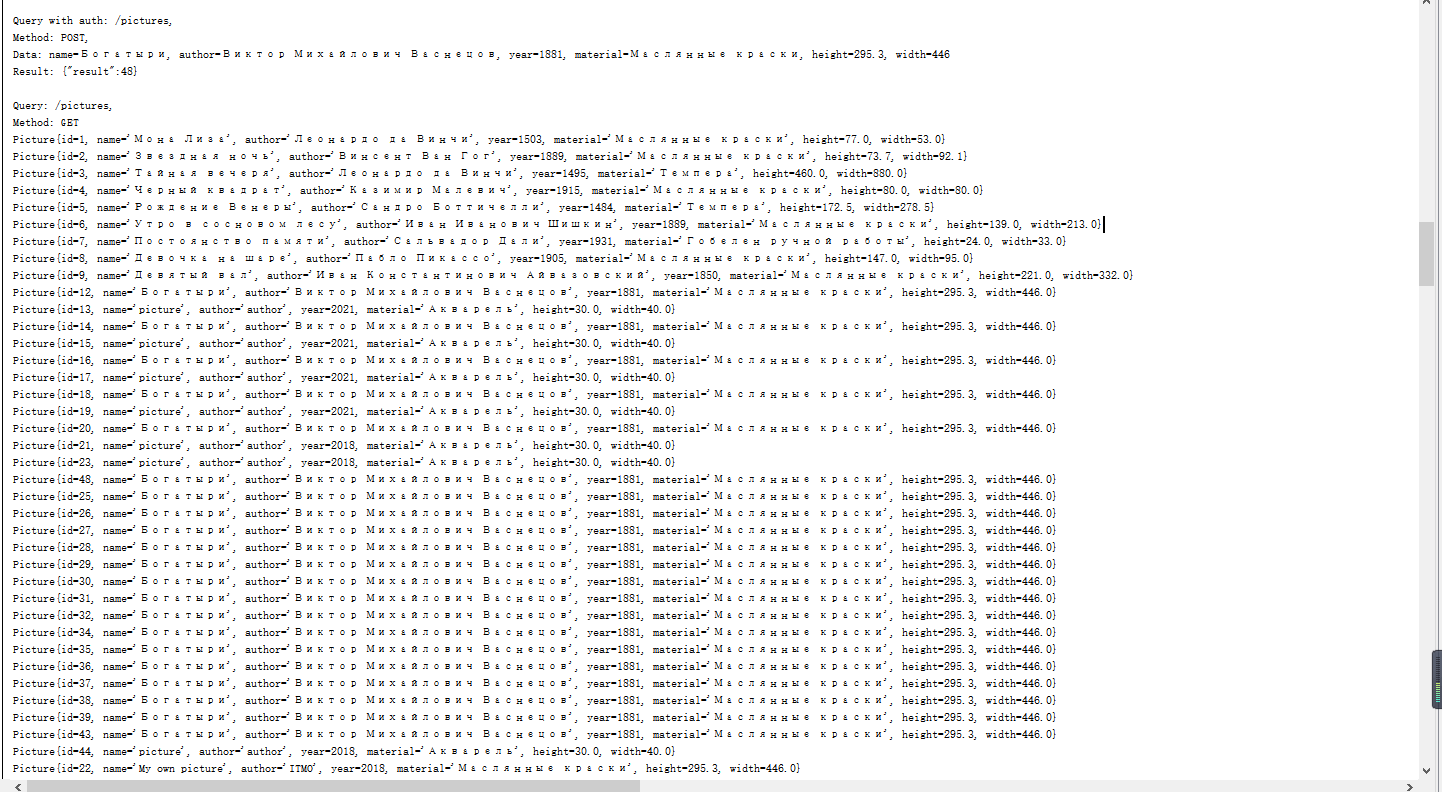
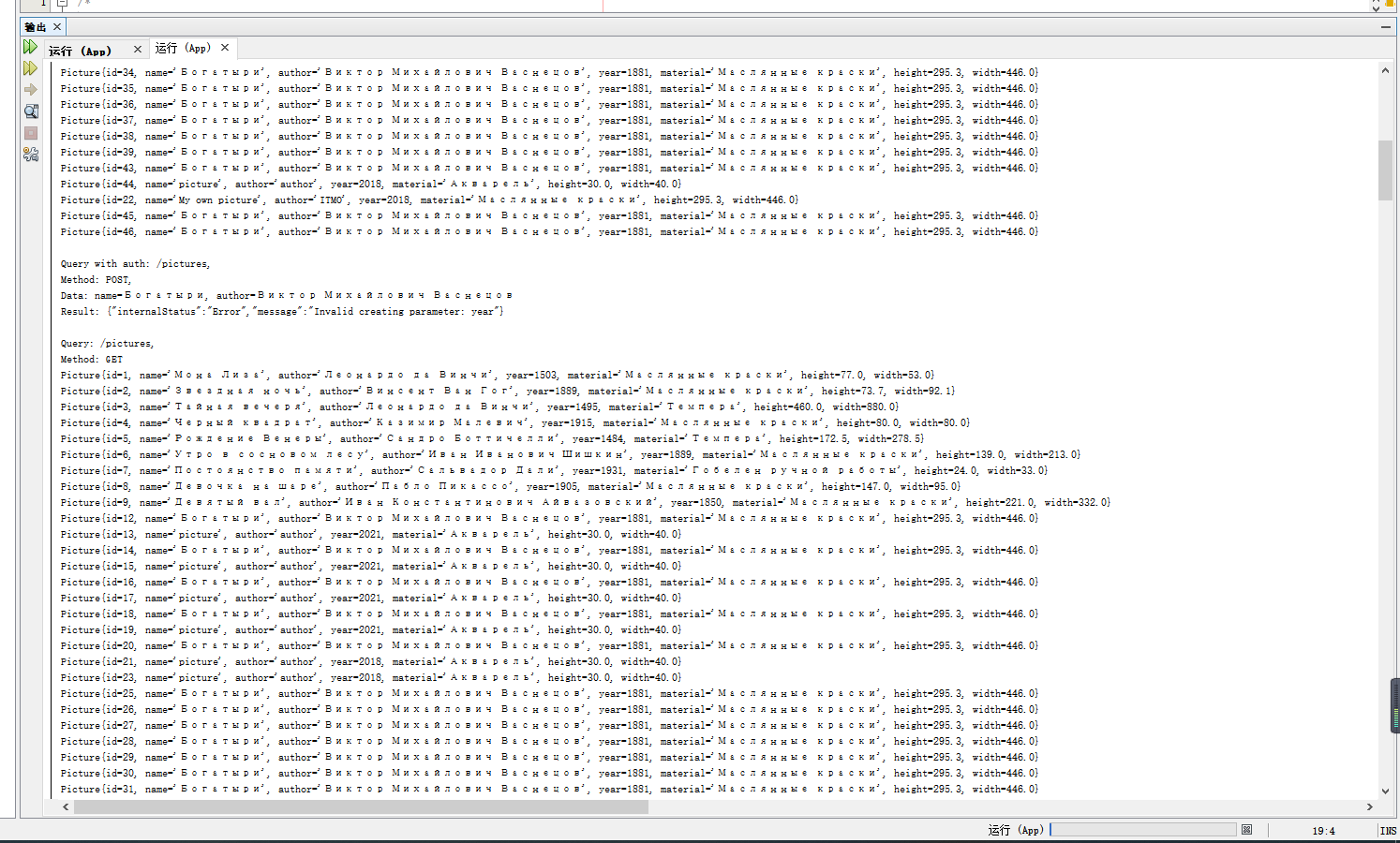
System.out.println(picture);

}

}

}

Рисунки 6.1



**Вывод:** в ходе выполнения работы, основываясь на информации из раздела 3.5, была добавлена обработка ошибок в сервис. Всоответствии с изменениями сервиса также было обновлено и клиентское приложение.