ДИСЦИПЛИНА **Алгоритмы и структуры данных с использованием компилируемых языков**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **Искусственного интеллекта**

КАФЕДРА **Технологий Искусственного Интеллекта**

(полное наименование кафедры)

ВИД УЧЕБНОГО МАТЕРИАЛА **Материалы для практических/семинарских занятий**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **Куликов Александр Анатольевич**

(фамилия, имя, отчество)

СЕМЕСТР **1, 2023-2024**

(указать семестр обучения, учебный год)

Практическая работа №5

Тема: Создание децентрализованной распределённой системы

Теоретическое введение

Споры о централизованных, децентрализованных и распределенных системах актуальны как для отдельных лиц, так и для организаций. Это затрагивает почти всех, кто пользуется Интернетом. Они лежат в основе развития и эволюции сетей, финансовых систем, компаний, приложений, веб-сервисов и многого другого.

Хотя все эти системы могут функционировать эффективно, некоторые из них по своей конструкции более стабильны и безопасны, чем другие. Системы могут быть очень маленькими, соединяя между собой всего несколько устройств и горстку пользователей. Или они могут быть огромными и охватывать страны и континенты. В любом случае они сталкиваются с одними и теми же проблемами: отказоустойчивость, затраты на обслуживание и масштабируемость.

Централизованные системы

В централизованной системе все пользователи подключены к центральному владельцу сети или «серверу». Центральный владелец хранит данные, к которым могут получить доступ другие пользователи, а также информацию о пользователях. Эта информация о пользователе может включать профили пользователей, пользовательский контент и многое другое. Централизованная система проста в установке и может быстро развиваться.

Но у этой системы есть важное ограничение. Если сервер выходит из строя, система перестает работать должным образом, и пользователи не могут получить доступ к данным. Поскольку централизованной системе необходим центральный владелец, для подключения всех других пользователей и устройств, доступность сети зависит от этого владельца. Добавьте к этому очевидные проблемы безопасности, которые возникают, когда один владелец хранит пользовательские

данные. Не сложно понять, почему централизованные системы больше не являются предпочтительным выбором для многих организаций.

Плюсы:

- Простое развертывание
- Быстрая разработка
- Доступность в обслуживании
- Практично, когда данные нужно контролировать централизованно

Минусы:

- Склонен к неудачам
- Повышенные риски безопасности и конфиденциальности для пользователей
- Более длительное время доступа к данным для пользователей, находящихся далеко от сервера

Децентрализованные системы

Как следует из названия, у децентрализованных систем нет единого центрального владельца. Вместо этого они используют нескольких центральных владельцев, каждый из которых обычно хранит копию ресурсов, к которым пользователи могут получить доступ.

Децентрализованная система может быть так же уязвима к сбоям, как и централизованная. Однако по своей конструкции система более устойчива к неисправностям. Это связано с тем, что при выходе из строя одного, нескольких центральных владельцев или серверов, другие продолжают работать и предоставляют пользователям доступ к данным.

Ресурсы остаются активными, если хотя бы один из центральных серверов продолжает работать. Обычно это означает, что владельцы системы могут ремонтировать неисправные серверы и решать любые другие проблемы, в то время как сама система продолжает работать в обычном режиме.

Сбои сервера в децентрализованной системе могут повлиять на производительность и ограничить доступ к некоторым данным. Но с точки зрения общего времени, безотказной работы эта система значительно лучше централизованной.

Еще одним преимуществом этой конструкции является то, что время доступа к данным часто меньше. Это потому, что владельцы могут создавать узлы в разных регионах или областях, где активность пользователей высока.

Однако децентрализованные системы по-прежнему подвержены тем же рискам безопасности и конфиденциальности для пользователей, что и централизованные системы. И хотя их отказоустойчивость выше, за это приходится платить. Поддержание децентрализованной системы обычно дороже.

Плюсы:

- Меньше вероятность отказа, чем в централизованной системе
- Лучшая производительность
- Позволяет создать более разнообразную и гибкую систему

Минусы:

- Риски безопасности и конфиденциальности для пользователей
- Более высокие затраты на обслуживание
- Непостоянная производительность при неправильной оптимизации
- Распределенная система

Распределенная система

Распределенная система похожа на децентрализованную в том, что у нее нет единого центрального владельца. Но если пойти дальше, то централизация исключается. В распределенной системе пользователи имеют равный доступ к данным, хотя при необходимости могут быть включены права пользователя. Лучшим примером обширной распределенной системы является сам Интернет.

Распределенная система позволяет пользователям делить владение данными. Аппаратные и программные ресурсы также распределяются между пользователями, что в некоторых случаях может улучшить производительность системы. Распределенная система защищена от независимого отказа компонентов, что может значительно улучшить ее время безотказной работы.

Распределенные системы развивались в результате ограничений других систем. В связи с растущими проблемами безопасности, хранения данных и конфиденциальности, а также постоянной потребностью в повышении производительности, распределенные системы становятся естественным выбором для многих организаций.

Поэтому неудивительно, что технологии, использующие распределенную систему, в первую очередь блокчейн, меняют многие отрасли.

Плюсы:

- Отказоустойчивой
- Прозрачный и безопасный
- Способствует совместному использованию ресурсов
- Чрезвычайно масштабируемый

Минусы:

- Сложнее развернуть
- Более высокие затраты на обслуживание

Чтобы понять, как работает базовая децентрализованная распределенная система, давайте посмотрим на пример.

Листинг 1 – Dockerfile

```
FROM maven:3.8.6-ibm-semeru-17-focal

ADD . /app
WORKDIR /app

RUN mvn clean install -DskipTests

FROM openjdk:17-oracle

LABEL name="ZEA"

ARG JAR_FILE=/app/target/*.jar
COPY --from=0 ${JAR_FILE} /application.jar
ENTRYPOINT ["java", "-jar", "application.jar"]
```

Содержание нашего docker-compose файла, в котором прописаны 4 экземпляра, которые создаются из одного образа.

Листинг 2 – docker-compose.yaml

```
version: "3.9"
services:
  app1:
    image: pr5-simple-app
    # image: evgenua/sem7-virt-documents:1.0
    # build:
    #   context: .
    #   dockerfile: Dockerfile
```

Продолжение листинга 2

```
    ports:
          - 8080:8080
        restart: on-failure
        environment:
          - NAME_APP=app1
        volumes:
          - ../upload-files:/upload-files

      app2:
        image: simple-app
        # build:
        #   context: .
        #   dockerfile: Dockerfile
        restart: on-failure
        # ports:
        #   - 8081:8080
        environment:
          - NAME_APP=app2
        volumes:
          - ../upload-files:/upload-files

      app3:
        image: simple-app
        # build:
        #   context: .
        #   dockerfile: Dockerfile
        # ports:
        #   - 8082:8080
        restart: on-failure
        environment:
          - NAME_APP=app3
        volumes:
          - ../upload-files:/upload-files

      app4:
        image: simple-app
        # build:
        #   context: .
        #   dockerfile: Dockerfile
        # ports:
        #   - 8083:8080
        restart: on-failure
        environment:
          - NAME_APP=app4
        volumes:
          - ../upload-files:/upload-files

      proxy:
        image: nginx:1.11
        ports:
          - '80:80'
        volumes:
          - ./nginx.conf:/etc/nginx/conf.d/default.conf:ro
```

Далее взглянем на конфигурационный файл nginx, который используется для конфигурации сервера: если сломается основной экземпляр, он будет пытаться пробросить экземпляры на остальные экземпляры.

Листинг 3 – nginx.conf

```
upstream myapp {
    server app1:8080;
    server app2:8080;
    server app3:8080;
    server app4:8080;
}

server {
    listen 80;

    location ^~ / {
        proxy_pass http://myapp;
    }
}
```

Также посмотрим файл FileService.java, в котором происходит обращение к другим классам и конструкторам системы, благодаря чему реализуется работа всей системы.

Листинг 4 – FileService.java

```
package com.example.simple.services;

import com.example.simple.dto.FileInfoDTO;
import com.example.simple.dto.FileUploadDTO;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import javax.annotation.PostConstruct;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.SimpleDateFormat;
import  java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.TimeZone;
```

Продолжение листинга 4

```
      import java.util.UUID;

      @Service
      @RequiredArgsConstructor
      public class FileService {
          @Value("${app.path.upload.file}")
          private String uploadPath;

          private SimpleDateFormat formatForDateNow = new SimpleDateFormat("hh:mm
dd.MM.yyyy");

          @PostConstruct
          public void init() {
              try {
                  Files.createDirectories(Paths.get(uploadPath));
              } catch (IOException e) {
                  throw new RuntimeException("Could not create upload folder!");
              }

               formatForDateNow.setTimeZone(TimeZone.getTimeZone("UTC"));
          }

          public void save(FileUploadDTO fileUploadDTO) {
              for(MultipartFile multipartFile: fileUploadDTO.getFiles()) {
                  String newFileName = UUID.randomUUID() + "_" +
multipartFile.getOriginalFilename();

                  saveFile(multipartFile,newFileName);
              }
          }

          public  List<FileInfoDTO>  getSentFiles()  {
              List<FileInfoDTO> fileInfoDTOList = new ArrayList<>();
              List<File> uploadFileList = getUploadFilesFromFolder();

              for (File file: uploadFileList) {
                  FileInfoDTO fileInfoDTO = new FileInfoDTO();

                  fileInfoDTO.setName(file.getName());
                  fileInfoDTO.setDate(formatForDateNow.format(new
Date(file.lastModified())));
                  fileInfoDTO.setUrl("/upload-files/" + file.getName());

                  fileInfoDTOList.add(fileInfoDTO);
              }
```

Продолжение листинга 4

```
            return fileInfoDTOList;
        }

        private void saveFile(MultipartFile file, String fileName) {
            try {
                Path root = Paths.get(uploadPath);
                if (!Files.exists(root)) {
                    init();
                }
                Files.copy(file.getInputStream(), root.resolve(fileName));
            } catch (Exception e) {
                throw new RuntimeException("Could not store the file. Error: " +
e.getMessage());
            }
        }

        private List<File> getUploadFilesFromFolder() {
            List<File> files = new ArrayList<>();
            File folder = new File(uploadPath);

            for (final File fileEntry : folder.listFiles()) {
                files.add(fileEntry);
            }

            return files;
        }
    }
```

Далее давайте посмотрим, как ведет себя приложение в запущенном виде.

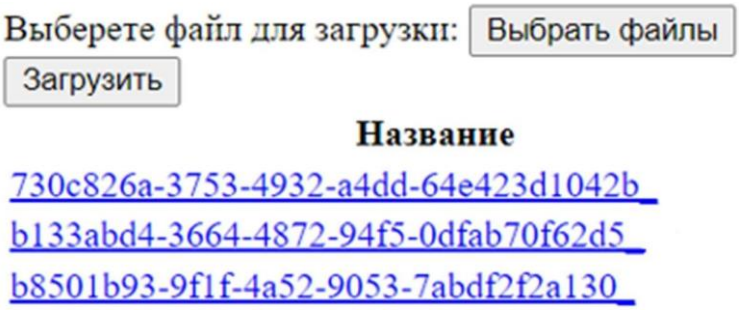

Рисунок 1 – Интерфейс приложения

Давайте попробуем добавить новый файл

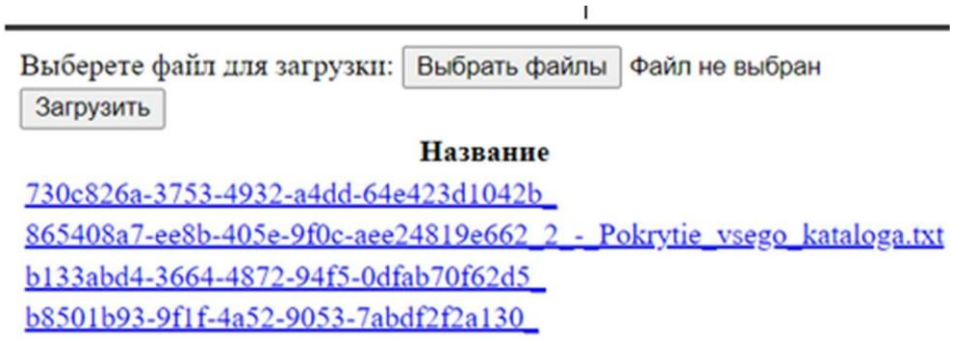

Рисунок 2 – Загрузили файл в систему

Далее посмотрим, будет ли работать система, если мы отключим основной экземпляр

Рисунок 3 – Выключаем основной экземпляр

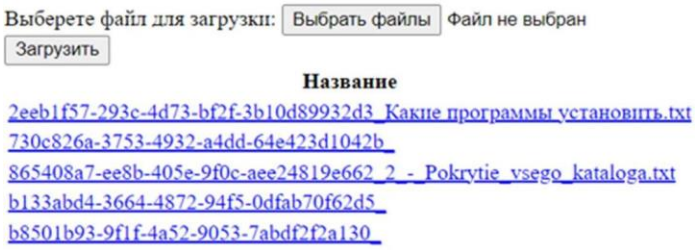

Рисунок 4 – Файл успешно загрузился

Далее давайте посмотрим, что будет, если отключить сразу два экземпляра.

Рисунок 5 – Выключаем еще один экземпляр

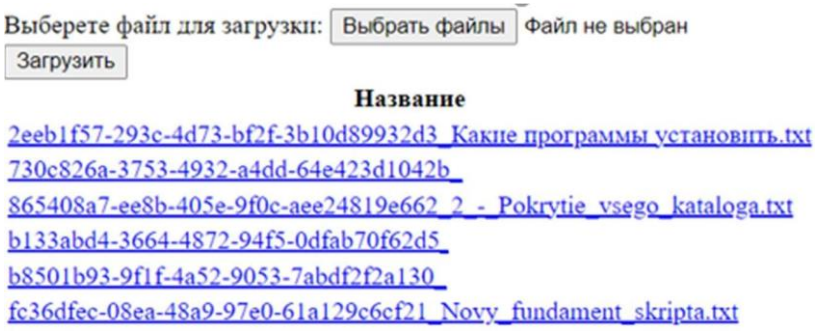

Рисунок 7 – Новый файл успешно загружен

Задание на практическую работу №5

Задание 1

Разработать децентрализованную распределенную систему по согласованной теме с преподавателем с использованием Spring/Spring Boot.

Требуемый функционал системы: загрузка и выгрузка файлов различных форматов, также допустим функционал по индивидуальному варианту.

Для демонстрации системы необходимо развернуть как минимум 4 экземпляра приложения при помощи Docker и продемонстрировать работоспособность при отключении каждого из 4х экземпляров, а также при отказе любых двух экземпляров.

В случае невозможности реализации системы с использованием предложенного стека технологий необходимо обосновать это и предложить решение на собственном стеке.