INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

CTESP Tecnologias Web e Dispositivos Móveis

Programação de Aplicação Desktop

Gestão de uma editora de Jogos(“MD”)

Elaborado por:

Dylan Fialho - 21135

Márcio Martins - 21136

Docentes:

David Fontes

Luís Rosário

Nuno Marques

12/01/2021

Índice

Índice de Figura

Introdução

Este trabalho descreve a criação de uma aplicação feita em JAVA no programa "*IntelliJ*" no âmbito da disciplina de Programação de Aplicação Desktop. Esta aplicação tem como objetivo a criação de uma aplicação que gere uma incubadora de outras empresas de jogos, através da inserção e remoção de dados, adição de novas empresas entre outros registos. Neste projeto vão ser utilizados principalmente dois programas, "*ItelliJ*" e "*phpMyAdmin*".

Organização do trabalho e ferramentas

Após se ver todos os temas, decidiu-se que não havia nenhum que fosse exatamente o desejado. Por isso, com a autorização do professor, decidiu-se criar um novo tema.
Após a criação do tema, e após se receber a aprovação do professor, fez-se a divisão de tarefas. Antes de se criar a base de dados, entre os membros, criou-se a mesma em papel, para ter a certeza que tudo funcionaria. Após isso usou-se o "*phpMyAdmin*" para a criar virtualmente. Depois da conclusão da base de dados, estava na altura de começar a parte da programação em JAVA. Começou-se com um simples login, e uma verificação se o utilizador é administrador ou não. Depois dessa verificação, criou-se os menus com as respetivas opções.

As ferramentas usadas neste projeto foram:
phpMyAdmin – Foi onde foi criada a base de dados;
XAMPP – programa usado para criar um servidor locado;
IntelliJ – IDE para JAVA.

Análise do problema

Com a criação desta aplicação o objetivo era simples. Quis-se que a aplicação pudesse ser usada por uma empresa que gerisse várias empresas mais pequenas de vídeo jogos.

Para isso foi preciso criar 2 tipos de menu. Um de administrador, que autoriza o manuseamento de empresas(mudar nomes, tipos de jogos, remover e adicionar), e um segundo menu de utilizador, que seria para verificações simples sobre os jogos.

Funcionalidades da aplicação

Menu de administrador:

- Inserir novos utilizadores;
- Inserir novas empresas adquiridas;
- Remover empresas;
- Alterar o nome de empresas.

Menu de utilizador:

- Ver os jogos;
- Consultar jogos por tipo;
- Consultar entre preços;
- Consultar entre datas.

Desenho da interface da aplicação

Menu de utilizador:

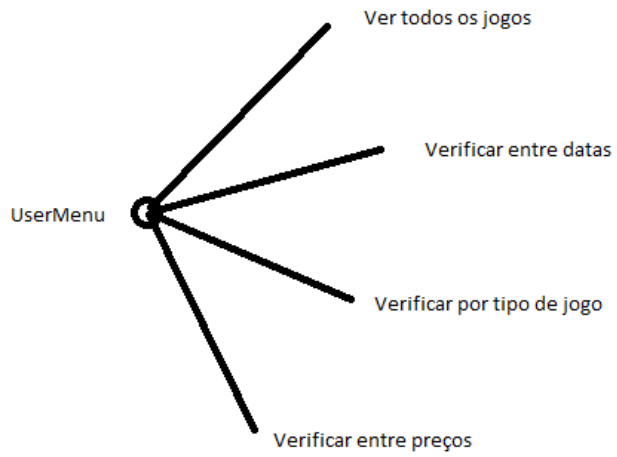

Figura 1 – Imagem do menu utilizador

Menu de administrador:

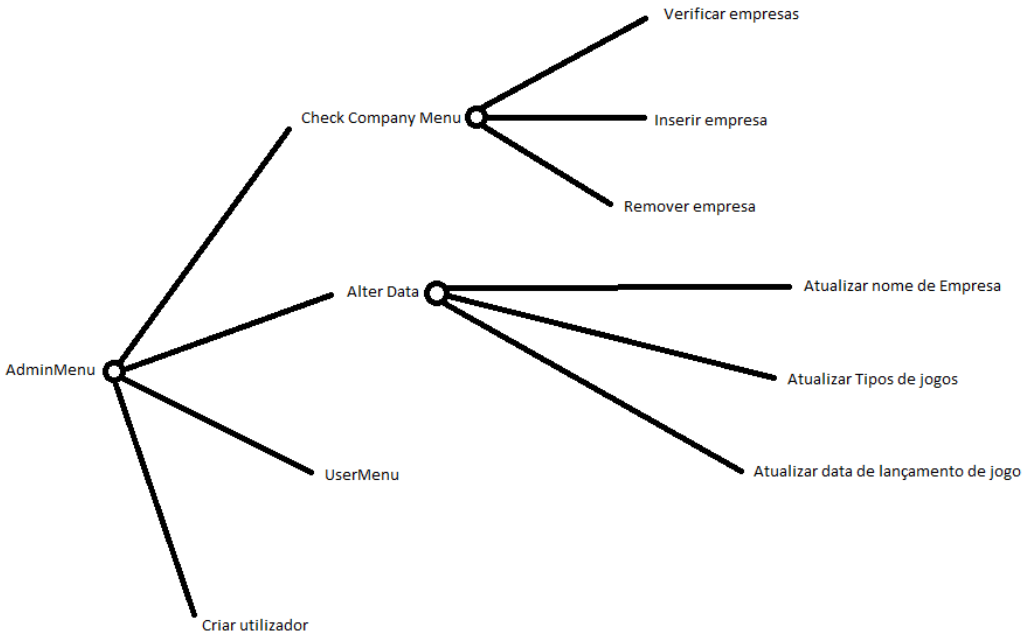

Figura 2 – Imagem do menu do administrador

Desenho da base de dados

Este foi o primeiro desenho da base de dados que foi feito durante uma aula de PAD.

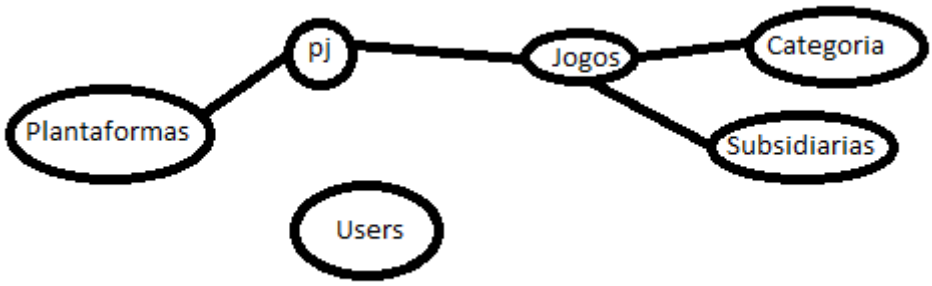

Figura 3 – Draft da base de dados

Conceção da base de dados

Esta base de dados está divida em cinco tabelas interligadas por chaves estrangeiras.

A primeira tabela “Subsidiária” foi criada com o ID de empresa, nome, tipo de jogos que a empresa cria (por exemplo: “*Origin*” que faz jogos de desporto, aventura e ação) e o seu site onde se pode ver a sua grande variedade de jogos.

Na tabela “Jogos” podemos encontrar o ID do jogo, o seu respetivo nome, o seu género, data de lançamento e o preço que vale. Criou-se assim uma tabela de relação com a tabela subsidiárias.

A tabela “Categoria” é composta pelo ID da categoria, o tipo de jogo, o ID do jogo e a plataforma.

A tabela “Plataformas” tem o ID de “Plataformas”, o nome e o seu site.

Para fazer a verificação dos utilizadores, criou-se uma tabela chamada “Users” onde vão estar todas as contas de utilizadores, com a verificação se é um administrador ou não.

Programação da lógica da aplicação

Este método efetua uma pergunta à base de dados e verifica se os inputs correspondem a um utilizador que está gravado, se estiver o programa continua, se não volta a repetir as perguntas.

```
public Login(DBConnection db, Scanner scanner) {
    boolean isValid;
    this.db = db;
    do {
        System.out.println("Enter User: ");
        String user = scanner.next();
        System.out.println("Password:");
        String pass = scanner.next();

        db.connect();
        List<Row> result = db.executeQuery(String.format("SELECT * FROM users WHERE nome_user =
'%s' " +
                "AND palavra_passe = '%s';", user, pass));
        db.close();

        if (result.size() > 0) {
            System.out.println(result.get(0));
            activeUser = new User(result.get(0));

            isValid = true;
        } else {
            System.out.println("Login Errado!");
            isValid = false;
        }
    } while (!isValid);
}
```

Com este método, chama-se o "*Login*" e verifica-se se o utilizador é administrador ou não. Dependendo, envia o utilizador para o menu "*User*" ou para o menu "*Admin*".

```
private static void login() {
    login = new Login(db, scanner);

    if (login.getActiveUser().getIsAdmin() > 0) {
        System.out.println("Bem vindo ao menu Administrador");
        adminMenu();
    } else {
        System.out.println("Bem vindo ao menu de Utilizador");
        userMenu();
    }
}
```

Este é o método do menu do utilizador. É através deste método que o utlizador pode fazer as suas pesquisas. Utilizando um loop "do while" consigo confirmar que o utilizador vai sempre inserir um número que esteja dentro das opções.

```
private static void userMenu() {
    int option = 0;
    do {
        System.out.println("Escolha a opção que quer:\n1. Consulta de todos os jogos\n2. Consulta entre datas\n3. " +
                "Consulta por tipos\n4. Consulta por preços\n5. Sair");
        option = scanner.nextInt();

        switch (option) {
            case 1:
                allComp("jogos");
                if (login.getActiveUser().getIsAdmin() != 0) {
                    adminMenu();
                } else {
                    userMenu();
                }
                break;
            case 2:
                checkBetweenDates();
                break;
            case 3:
                checkGameType();
                break;
            case 4:
                checkByGamePrice();
                break;
            case 5:
                System.exit(0);
        }
    } while (option < 0 || option > 6);
}
```

Este é o método do menu do administrador. Em termos da sua implementação é muito parecido ao anterior. A única diferença é as suas opções.

```
private static void adminMenu() {
    int option = 0;

    do {
        System.out.println("Bem vindo ao menu de administradores\n1. Consulta de Subsidiárias\n2. Alteração de dados\n" +
                "3. Aceder a Menu de utilizadores\n4. Sair");
        option = scanner.nextInt();

        switch (option) {
            case 1:
                checkCompMenu();
                break;
            case 2:
                alterAllData();
                break;
            case 3:
                userMenu();
                break;
            case 4:
```

```
            createUser();
        case 5:
            System.exit(0);
    }
  } while (option < 0 || option > 6);
}
```

Este método recebe uma "*String*" como parâmetro que será o nome da tabela que se quer consulta. Depois obtém a lista completa de dados da base de dados e imprime cada linha dessa mesma lista.

```
private static List<Row> allComp(String tableName) {
  db.connect();
  List<Row> result = db.executeQuery(String.format("SELECT * FROM %s;", tableName));
  db.close();

  for (Row r :
       result) {
    System.out.println(r.toString());
  }
  return result;
}
```

Este método pede 2 datas ao utilizador e pede a base de dados para mostrar todos os resultados entres essas datas.

```
private static void checkBetweenDates() {
  System.out.println("Qual é a primeira data? (YYYY-MM-DD)");
  String firstDate = scanner.next();
  System.out.println("Qual é a segunda data? (YYYY-MM-DD)");
  String secondDate = scanner.next();

  db.connect();
  List<Row> result = db.executeQuery(String.format("SELECT * FROM jogos WHERE
data_lancamento BETWEEN CAST('%s' AS DATE) AND CAST('%s' AS DATE);",
      firstDate, secondDate));
  db.close();

  for (Row r :
       result) {
    System.out.println(r.toString());
  }

  if (login.getActiveUser().getIsAdmin() != 0) {
    adminMenu();
  } else {
    userMenu();
  }
}
```

Este método pede ao utilizador para inserir o tipo de jogo que quer ver, e verifica na base de dados quais os jogos com esse tipo.

```
private static void checkGameType() {
    System.out.println("Qual é o tipo de jogo?");
    String gameType = scanner.next();

    db.connect();
    List<Row> result = db.executeQuery(String.format("SELECT * FROM jogos WHERE tipo_jogo
LIKE '%%%s%%'", gameType));
    db.close();

    for (Row r :
            result) {
        System.out.println(r.toString());
    }

    if (login.getActiveUser().getIsAdmin() != 0) {
        adminMenu();
    } else {
        userMenu();
    }
}
```

Este método pede 2 preços ao utilizador, e depois verifica na base de dados todos os jogos com preços entre os 2 inputs.

```
private static void checkByGamePrice() {
    System.out.println("Qual é o range de preço dos jogos?\nPreço mais baixo:");
    int lowGamePrice = scanner.nextInt();
    System.out.println("\nPreço mais alto:");
    int highGamePrice = scanner.nextInt();

    db.connect();
    List<Row> result = db.executeQuery(String.format("SELECT * FROM jogos WHERE preco > %s
AND preco < %s", lowGamePrice,
            highGamePrice));
    db.close();

    for (Row r :
            result) {
        System.out.println(r.toString());
    }

    if (login.getActiveUser().getIsAdmin() != 0) {
        adminMenu();
    } else {
        userMenu();
    }
}
```

Este método pede ao utilizador para escolher uma consola e mostra todos os jogos disponíveis para essa mesma.

```
private static void checkByConsole(){
    int option = 0;
    db.connect();

    do{
        System.out.println("Qual é a consola?\n1. PlayStation\n2.
Xbox\n3. Nintendo\n4. Computador\n5. Voltar ao menu");
        option = scanner.nextInt();
        List<Row> result;

        switch (option){
            case 1:
                result = db.executeQuery("SELECT * FROM ((plantaformas
INNER JOIN pj ON plantaformas.id_plantaformas = pj." +
                        "id_plantaforma) INNER JOIN jogos ON
pj.id_jogo = jogos.id_jogo) WHERE plantaformas.nome =
'Playstation';");
                db.close();

                for (Row r :
                        result) {
                    System.out.println(r.toString());
                }
                break;
            case 2:
                result = db.executeQuery("SELECT * FROM ((plantaformas
INNER JOIN pj ON plantaformas.id_plantaformas = pj." +
                        "id_plantaforma) INNER JOIN jogos ON
pj.id_jogo = jogos.id_jogo) WHERE plantaformas.nome = 'Xbox';");
                db.close();

                for (Row r :
                        result) {
                    System.out.println(r.toString());
                }
                break;
            case 3:
                result = db.executeQuery("SELECT * FROM ((plantaformas
INNER JOIN pj ON plantaformas.id_plantaformas = pj." +
                        "id_plantaforma) INNER JOIN jogos ON
pj.id_jogo = jogos.id_jogo) WHERE plantaformas.nome = 'Nintendo';");
                db.close();

                for (Row r :
                        result) {
                    System.out.println(r.toString());
                }
                break;
            case 4:
                result = db.executeQuery("SELECT * FROM ((plantaformas
INNER JOIN pj ON plantaformas.id_plantaformas = pj." +
                        "id_plantaforma) INNER JOIN jogos ON
pj.id_jogo = jogos.id_jogo) WHERE plantaformas.nome = 'Computador';");
                db.close();

                for (Row r :
                        result) {
                    System.out.println(r.toString());
```

```
                }
                break;
            case 5:
                if (login.getActiveUser().getIsAdmin() != 0) {
                    adminMenu();
                } else {
                    userMenu();
                }
                break;
        }
    }while(option < 0 || option > 5);

    if (login.getActiveUser().getIsAdmin() != 0) {
        adminMenu();
    } else {
        userMenu();
    }
}
```

Isto é um segundo menu que é acedido pelo menu de administradores. É o menu para verificar, inserir ou apagar empresas.

```
private static void checkCompMenu() {
   int option = 0;

   do {
      System.out.println("Escolha uma das opções:\n1. Verificar Empresas\n2. Inserir Empresas\n3.
Remover Empresas");
      option = scanner.nextInt();

      switch (option) {
         case 1:
            allComp("subsidiarias");
            adminMenu();
            break;
         case 2:
            insComp();
            break;
         case 3:
            remComp();
            break;
      }
   } while (option > 3 || option < 0);

   adminMenu();
}
```

Com este método é possível inserir uma nova empresa na base de dados.

```
private static void insComp() {
   System.out.println("Qual o nome da subsidiária:");
   String name = scanner.next();
   System.out.println("Qual o tipo de jogos:");
   String typeGames = scanner.next();
   System.out.println("Link da subsidiária:");
   String link = scanner.next();

   db.connect();
   db.executeInsert(String.format("INSERT INTO subsidiarias(nome_empresa, tipo_de_jogo,
```

```
site_empresa) VALUES('%s', '%s', '%s')",
        name, typeGames, link));
    db.close();
}
```

Este método serve para apagar uma empresa da base de dados.

```
private static void remComp() {
    List<Row> result = allComp("subsidiarias");
    List<Integer> listIds = listIds(result, 0);

    System.out.println("Qual a empresa que quer remover?");
    int option = scanner.nextInt();

    if (listIds.contains(option)) {
        db.connect();
        db.executeInsert(String.format("DELETE FROM subsidiarias WHERE id_empresa = %d", option));
        db.close();
        System.out.println("A empresa foi apagada corretamente!");
    } else {
        System.out.println("Opção não suportada!");
    }
    adminMenu();
}
```

Este método foi criado para alterar os dados de uma certa empresa, como o nome, os tipos de jogos e as datas de lançamento de um certo jogo. Para isso o método pergunta qual é a empresa que quer alterar.

```
private static void alterAllData() {
    List<Row> result = allComp("subsidiarias");
    int option = 0;

    do {
        System.out.println("Qual das subsidiárias quer alterar?");
        option = scanner.nextInt();
        if (listIds(result, 0).contains(option)) {
            System.out.println("O que é que quer alterar?\n1. Nome da empresa\n2. Tipos de jogos\n3. " +
                    "Data de lançamento\n4. Voltar ao menu");
            int optionSubMenu = scanner.nextInt();
            db.connect();

            switch (optionSubMenu) {
                case 1:
                    updateCompName(option);
                    break;
                case 2:
                    updateGameType(option);
                    break;
                case 3:
                    alterDate(option);
                    break;
                case 4:
                    adminMenu();
            }
        }
    } while (option < 0 || !listIds(result, 0).contains(option));
    adminMenu();
}
```

Este método recebe a opção da empresa obtida no menu anterior, e executa a mudança para o novo nome.

```
private static void updateCompName(int option) {
    System.out.println("Qual é o novo nome?");
    String newName = scanner.next();
    db.executeInsert(String.format("UPDATE subsidiarias SET nome_empresa = '%s' WHERE
id_empresa = %d;",
            newName, option));
    db.close();
}
```

Este método recebe a opção da empresa obtida no menu anterior, e executa a mudança para o novo tipo de jogo.

```
private static void updateGameType(int option) {
    System.out.println("Qual é o novo tipo de jogo? (Pode inserir mais que um separado por '/')");
    String newGameType = scanner.next();
    db.executeInsert(String.format("UPDATE subsidiarias SET tipo_de_jogo = '%s' WHERE id_empresa
= %d",
            newGameType, option));
    db.close();
}
```

Este método mostra uma lista de todos os jogos que estão em desenvolvimento pela empresa escolhida anteriormente, e pergunta qual é a nova data de lançamento. Depois atualiza os novos dados na base de dados.

```
private static void alterDate(int option) {
    List<Row> tempResult = db.executeQuery(String.format("SELECT * FROM jogos WHERE
id_empresa = %d",
            option));
    for (Row row : tempResult) {
        System.out.println(row.toString());
    }
    System.out.println("Qual o id do jogo que quer alterar?");
    int tempOption = scanner.nextInt();
    if (listIds(tempResult, 0).contains(tempOption)) {
        System.out.println("Para que data quer alterar? (YYYY-MM-DD)");
        String newDate = scanner.next();
        db.executeInsert(String.format("UPDATE jogos SET data_lancamento = '%s' WHERE id_jogo =
%d",
                newDate, tempOption));
    } else {
        System.out.println("Essa opção não existe!");
    }
}
```

Este método é o método para criar novos utilizadores. Apenas recebe os dados e, se esse user ainda não existir, envia para a base de dados.

```
private static void createUser(){
    System.out.println("Insira um nome de utilizador:");
    String userName = scanner.next();
    System.out.println("Insira o nome do utilizador:");
    String name = scanner.next();
    System.out.println("Insira o email do utilizador:");
    String email = scanner.next();
    System.out.println("Insira uma palavra-passe para o utilizador:");
    String password = scanner.next();
    System.out.println("O utilizador é admin? (1- Sim, 0 - Não)");
    int isAdmin = scanner.nextInt();

    db.connect();
    List<Row> result = db.executeQuery(String.format("SELECT * FROM users WHERE nome_user =
%s AND email = %s;",
            userName, email));

    if(result.size() != 0){
        System.out.println("Este user ja existe");
    }else {
        db.executeInsert(String.format("INSERT INTO users (nome_user, nome, palavra_passe, isAdmin)
VALUES('%s','%s','%s'" +
                "','%s', %d);", userName, name, email, password, isAdmin));
        db.close();
    }
}
```

Este método recebe uma lista e uma posição para o "*array*" da lista e devolve todos os campos dessa mesma posição numa lista.

```
private static List<Integer> listIds(List<Row> result, int position) {
    List<Integer> listId = new ArrayList<>();

    for (Row countComp : result) {
        listId.add(Integer.parseInt(countComp.getColumns().get(position)));
    }

    return listId;
}
```

Conclusão

Neste projeto criamos uma empresa que vende jogos de outras subsidiárias, onde criamos dois níveis de visualização/acesso (Menu User / Menu Admin). Para se poder visualizar os jogos tivemos de criar uma base de dados onde se colocou as subsidiárias, as plataformas, os jogos, as categorias e os users.

Foi possível alcançar todos os objetivos traçados no início do projeto, com a adição de mais algumas funções.

Webgrafia

https://docs.oracle.com/javase/7/docs/api/java/lang/package-summary.html

https://dev.mysql.com/doc/

https://stackoverflow.com/