Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

МФКТиУ, ФПИиКТ, СППО

Лабораторная работа №7

по дисциплине

«Программирование»

Вариант - 311303

Выполнил:

Анищенко Анатолий Алексеевич

Группа: PЗ112

Санкт-Петербург

2019 г.

**Задание:** доработать программу из лабораторной работы №6 следующим образом:

1. В класс, объекты которого хранятся в коллекции, добавить поле типа java.time.ZonedDateTime, в котором должны храниться дата и время создания объекта.
2. Обеспечить возможность регистрации и авторизации пользователей.
3. При регистрации генерировать случайный пароль и отправлять его на почту, указанную при регистрации.
4. Для отправки почтовых уведомлений использовать JavaMail API.
5. Пароли при хранении хэшировать алгоритмом MD2.
6. Реализовать ассоциацию между объектом из коллекции и пользователем, его создавшим. Пользователи могут просматривать объекты, созданные всеми пользователями, а модифицировать - только свои.
7. Для идентификации пользователя отправлять логин и пароль с каждым запросом.
8. Обеспечить хранение всех данных (объектов коллекции, пользователей и ассоциаций) в реляционной базе данных

Порядок выполнения работы:

* В качестве базы данных использовать PostgreSQL.
* Для подключения к БД на кафедральном сервере использовать хост pg, имя базы данных - studs, имя пользователя/пароль совпадают с таковыми для подключения к серверу.
* Данные для подключения к почтовому серверу уточняются.
* Отчёт по работе должен содержать:

Текст задания.

* Диаграмма классов разработанной программы.
* Исходный код программы.
* Выводы по работе.

**Код программы:**

<https://github.com/AnatoliiAnishchenko/ITMO/tree/master/Programming/Lab7>

**EmailSendler.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | **package** server;  **import** **javax.mail.\***;  **import** **javax.mail.internet.InternetAddress**;  **import** **javax.mail.internet.MimeMessage**;  **import** **java.util.Properties**;  **public** **class** **EmailSendler** {  **private** **static** **final** String FROM = "ttaaa.ForApplication@gmail.com";  **private** **static** **final** String PASSWORD = "password";  **private** **static** Session session;  **public** **EmailSendler**() {  Properties prop = **new** Properties();  prop.put("mail.smtp.host", "smtp.gmail.com");  prop.put("mail.smtp.port", "587");  prop.put("mail.smtp.auth", "true");  prop.put("mail.smtp.starttls.enable", "true");  session = Session.getInstance(prop,  **new** javax.mail.Authenticator() {  **protected** PasswordAuthentication **getPasswordAuthentication**() {  **return** **new** **PasswordAuthentication**(FROM, PASSWORD);  }  });  }  **public** **static** **void** **sendMail**(String TO, String subject, String text) {  **try** {  Message message = **new** MimeMessage(session);  message.setFrom(**new** InternetAddress(FROM));  message.setRecipients(  Message.RecipientType.TO,  InternetAddress.parse(TO)  );  message.setSubject(subject);  message.setText(text + "\n\nWith best regards,\nMoomin's application development team");  Transport.send(message);  } **catch** (MessagingException e) {  e.printStackTrace();  }  }  **public** **static** **void** **sendMail**(String TO, String text) {  sendMail(TO, "Moomin's application", text);  }  } |

**StringRandomizer.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **package** server;  **import** **java.util.Random**;  **public** **class** **StringRandomizer** {  **private** **static** **final** Random RANDOM = **new** Random();  **private** **static** **final** String ALPHABET = "@#%$0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";  **public** **static** String **randomString**(**int** length) {  StringBuilder returnValue = **new** StringBuilder(length);  **for** (**int** i = **0**; i < length; i++) {  returnValue.append(ALPHABET.charAt(RANDOM.nextInt(ALPHABET.length())));  }  **return** **new** **String**(returnValue);  }  **public** **static** String **randomPassword**() {  **return** **randomString**(**10**);  }  } |

**PasswordEncryptor.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | **package** server;  **import** **java.math.BigInteger**;  **import** **java.security.MessageDigest**;  **import** **java.security.NoSuchAlgorithmException**;  **public** **class** **PasswordEncryptor** {  **private** **static** **final** String salt = "\*\*\*\*\*\*\*";  **private** **static** **final** String pepper = "\*\*\*\*\*\*\*";  **public** **static** String **encrypt**(String password) {  String input = pepper + password + salt;  **try** {  // getInstance() method is called with algorithm MD2  MessageDigest md = MessageDigest.getInstance("MD2");  // digest() method is called  // to calculate message digest of the input string  // returned as array of byte  **byte**[] messageDigest = md.digest(input.getBytes());  // Convert byte array into signum representation  BigInteger no = **new** BigInteger(**1**, messageDigest);  // Convert message digest into hex value  String hashtext = no.toString(**16**);  // Add preceding 0s to make it 32 bit  **while** (hashtext.length() < **32**) {  hashtext = "0" + hashtext;  }  // return the HashText  **return** hashtext;  }  // For specifying wrong message digest algorithms  **catch** (NoSuchAlgorithmException e) {  **throw** **new** **RuntimeException**(e);  }  }  } |

**DBConnector.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183 | **package** server;  **import** **moominClasses.Field**;  **import** **moominClasses.Moomin**;  **import** **java.sql.\***;  **import** **java.time.ZoneId**;  **import** **java.time.ZonedDateTime**;  **import** **java.time.format.DateTimeFormatter**;  **import** **java.util.Vector**;  **public** **class** **DBConnector** {  **private** **static** **final** String DB\_DRIVER = "org.postgresql.Driver";  **private** **static** **final** String DB\_CONNECTION = "jdbc:postgresql://localhost:9080/studs";  **private** **static** Connection dbConnection;  **public** **DBConnector**() {  **try** {  Class.forName(DB\_DRIVER);  String login = "s264448";  String password = "eli307";  dbConnection = DriverManager.getConnection(DB\_CONNECTION, login, password);  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  } **catch** (SQLException e) {  e.printStackTrace();  }  }  **public** **static** **boolean** **newUser**(String username, String email, String password) {  **try** {  ResultSet rs = dbRequest("SELECT user\_name, email FROM users");  **while** (rs.next()) {  **if** (rs.getString(**1**).equals(username) || rs.getString(**2**).equals(email)) {  **return** **false**;  }  }  } **catch** (SQLException e) {  e.printStackTrace();  }  String query = "INSERT INTO users(user\_name, password, email) VALUES ";  **try** (Statement stmt = dbConnection.createStatement()){  stmt.executeUpdate(query + "('" + username + "', '" + PasswordEncryptor.encrypt(password) + "', '" + email + "')");  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** **true**;  }  **public** **static** ResultSet **dbRequest**(String request) **throws** SQLException {  Statement stmt = dbConnection.createStatement();  ResultSet res = stmt.executeQuery(request);  **return** res;  }  **public** **static** **boolean** **checkPassword**(String username, String password) {  **boolean** res = **false**;  String encryptedPassword = PasswordEncryptor.encrypt(password);  **try** {  ResultSet rs = dbRequest("SELECT password FROM users WHERE user\_name='" + username + "'");  **while** (rs.next()) {  **if** (rs.getString(**1**).equals(encryptedPassword)) {  res = **true**;  }  }  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** res;  }  **public** **static** **int** **getUserID**(String username) {  **int** res = -**1**;  **try** {  ResultSet rs = dbRequest("SELECT user\_id FROM users WHERE user\_name='" + username + "'");  **while** (rs.next()) {  res = rs.getInt(**1**);  }  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** res;  }  **public** **static** **boolean** **addAllMooomins**(Vector<Moomin> moomins, **int** userID) {  String query = "INSERT INTO moomins(owner\_id, creation\_time, moomin\_name, moomin\_field, moomin\_type, " +  "moomin\_gender, moomin\_position, moomin\_condition) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";  **boolean** flag = **true**;  **for** (**int** i = **0**; i < moomins.size(); i++) {  Moomin moomin = moomins.get(i);  **try** (PreparedStatement stmt = dbConnection.prepareStatement(query)) {  stmt.setInt(**1**, userID);  stmt.setTimestamp(**2**, java.sql.Timestamp.valueOf(moomin.getCreationTime().toLocalDateTime()));  stmt.setString(**3**, moomin.getName());  stmt.setString(**4**, moomin.getField().toString());  stmt.setString(**5**, moomin.getType().toString());  stmt.setString(**6**, moomin.getGender().toString());  stmt.setString(**7**, moomin.getPosition().toString());  stmt.setString(**8**, moomin.getCondition().toString());  stmt.executeUpdate();  } **catch** (SQLException e) {  e.printStackTrace();  flag = **false**;  }  }  **return** flag;  }  **public** **static** Vector<Moomin> **getMoomins**() {  Vector<Moomin> res = **new** Vector<>();  **try** {  ResultSet rs = dbRequest("SELECT \* FROM moomins");  **while** (rs.next()) {  ZonedDateTime time = ZonedDateTime.of(rs.getTimestamp(**2**).toLocalDateTime(), ZoneId.of("Europe/Moscow"));  String name = rs.getString(**3**);  Field field = **new** Field(rs.getString(**4**));  Moomin.TypeOfMoomin type = Moomin.TypeOfMoomin.strToType(rs.getString(**5**));  Moomin.Gender gender = Moomin.Gender.strToGender(rs.getString(**6**));  Moomin.Position position = Moomin.Position.strToPosition(rs.getString(**7**));  Moomin.Condition condition = Moomin.Condition.strToCondition(rs.getString(**8**));  res.add(**new** Moomin(time, name, field, type, gender, position, condition));  }  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** res;  }  **public** **static** **int** **getCountOfMoomins**() {  **int** count = **0**;  **try** {  ResultSet res = dbRequest("SELECT COUNT(\*) FROM moomins");  **while** (res.next()) {  count = res.getInt(**1**);  }  } **catch** (SQLException e) {  e.printStackTrace();  }  **return** count;  }  **public** **static** **void** **clearMoomins**(**int** userID) {  String query = "DELETE \* FROM moomins WHERE owner\_id=" + userID;  **try** (Statement stmt = dbConnection.createStatement()){  stmt.executeUpdate(query);  } **catch** (SQLException e) {  e.printStackTrace();  }  }  **public** **static** **boolean** **remove**(Moomin moomin, **int** userID) {  String query = "DELETE \* FROM moomins WHERE owner\_id=" + userID + "AND moomin\_name='" + moomin.getName() + "'";  **try** (Statement stmt = dbConnection.createStatement()){  stmt.executeUpdate(query);  **return** **true**;  } **catch** (SQLException e) {  e.printStackTrace();  **return** **false**;  }  }  **public** **static** **boolean** **removeLowerest**(Moomin moomin, **int** userID) {  String query = "DELETE \* FROM moomins WHERE owner\_id=" + userID + "AND moomin\_name<='" + moomin.getName() + "'";  **try** (Statement stmt = dbConnection.createStatement()){  stmt.executeUpdate(query);  **return** **true**;  } **catch** (SQLException e) {  e.printStackTrace();  **return** **false**;  }  }  } |

**Tunnel.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96 | **package** server;  **import** **com.jcraft.jsch.\***;  **import** **java.net.InetAddress**;  **import** **java.net.UnknownHostException**;  **import** **java.sql.\***;  **public** **class** **Tunnel** {  **private** String host;  **private** String user;  **private** String password;  **private** **int** port;  **private** String tunnelRemoteHost;  **private** **int** tunnelLocalPort;  **private** **int** tunnelRemotePort;  /\*\*  \* SSH Tunnel constructor  \*  \* @param host Target SSH host, which you will be connected (example: "helios.se.ifmo.ru")  \* @param user User on target SSH host (example: s244444)  \* @param password Password on target SSH host  \* @param port Port on target SSH host (example: 2222)  \* @param tunnelRemoteHost The host from server, which you want to connect (example: "pg")  \* @param tunnelLocalPort The port on your machine (example: 5454)  \* @param tunnelRemotePort The port on SSH server (example: 5432)  \*/  **public** **Tunnel**(String host,  String user,  String password,  **int** port,  String tunnelRemoteHost,  **int** tunnelLocalPort,  **int** tunnelRemotePort) {  **this**.user = user;  **this**.password = password;  **this**.port = port;  **this**.host = host;  **try** {  InetAddress.getByName(host);  } **catch** (UnknownHostException e) {  System.out.println("Uncorrect SSH host");  e.printStackTrace();  }  **this**.tunnelRemotePort = tunnelRemotePort;  **this**.tunnelRemoteHost = tunnelRemoteHost;  **this**.tunnelLocalPort = tunnelLocalPort;  }  /\*\*  \* Establish SSH connection and set port forwarding L  \* @return assigned port from connection  \*/  **public** **int** **connect**() {  JSch jsch = **new** JSch();  **try** {  Session session = jsch.getSession(user, host, port);  session.setPassword(password);  localUserInfo lui = **new** localUserInfo();  session.setUserInfo(lui);  //CONNECT SSH  System.out.println("Establishing SSH connection to " +  host + " : " + port);  session.setConfig("StrictHostKeyChecking", "no");  session.connect(**3000**);  //DO PORT FORWARDING  **int** assingedPort = session.setPortForwardingL(tunnelLocalPort, tunnelRemoteHost, tunnelRemotePort);  System.out.println("Connected");  System.out.println("localhost:" + assingedPort + " -> " + tunnelRemoteHost + ":" + tunnelRemotePort);  **return** assingedPort;  } **catch** (JSchException e) {  System.out.println("SSH connection error");  e.printStackTrace();  }  System.exit(-**1**);  **return** -**1**;  }  **class** **localUserInfo** **implements** UserInfo {  String passwd;  **public** String **getPassword**() { **return** passwd; }  **public** **boolean** **promptYesNo**(String str) { **return** **true**; }  **public** String **getPassphrase**() { **return** **null**; }  **public** **boolean** **promptPassphrase**(String message) { **return** **true**; }  **public** **boolean** **promptPassword**(String message) { **return** **true**; }  **public** **void** **showMessage**(String message) { }  }  } |

**Выводы:** в процессе выполнения лабораторной работы были получены навыки работы с Java Mail API, а также навыки работы с базами данных, на примере PostgreSQL.