Roskilde Daycare Project

KEA Copenhagen School of Design and Technology

Lygten 37 - 2200 København N

Computer Science, 2nd Semester 2020

Group 8 : Remi Foss, Aleksandar Miroslavov Minchev, Maja Rebeka Miskéri, Dagmara Przygocka, Cristian-Valentin Purcea

Table of contents

Project Overview

Introduction

In this project our team created administrative IT-system to for Roskilde Daycare. The main purpose of the system is to aid in maintaining the daycare's administrative tasks in a simple manner.

Some parts of the code & functionality will be presented and explained here, although, for a better comprehension of the project, we recommend checking out the GitHub Repository :

- https://github.com/CristiPV/Roskilde-Daycare-Project.git

Requirements

Here is a list of the functional requirements we discussed at the beginning of the project :

Requirements

- Proper User's permissions management;
- Better support for administrative tasks :
  - creating & managing work schedules;
  - managing children's data;
  - managing parents' data;
  - managing telephone list;
  - managing an appointment list;
  - managing a waiting list;
- User-friendly.

Later on, we conducted a deeper analysis of the projects requirements and we created a FURPS+ Model, which can be found in the Appendices section, or by clicking here.

System description:

Available features:

- add, remove, display, child
- add, remove, display teacher
- add, display parent
- add, delete, display schedule
- add activity to the schedule
- add, display activities
- add, delete, display group,
- add child to the group,
- add, delete, display waiting list
- add, delete, display invoice
- add, delete, display appointment

Data storage:

All the data used by the System is being stored on an online Database hosted by the Amazon Web Service platform.

Description:

In order to create the system we used Java programming language to create functionalities and MySQL to deliver necessary information from database.

In the system there are two types of users:

-Administrator : there is only one Administrator account which has access to all functionality of the system.

-Teacher : there is several accounts of Teacher user and more can be created by the administrator.

Teacher account has access to functionalities like:

- add, delete, display waiting list,
- add, delete, display appointments,
- add, delete, display groups,
- display schedule,
- display phone list

Feasibility Study

1. *Operational Feasibility*:
   - The system complexity is quite low, allowing the potential users to access key functionality it with ease.
   - The project will aim to solve the business' problems by reducing the time required to maintain administrative tasks.
2. *Technical Feasibility*:
   - The team will be able to make the system work.
   - The technology required for the completion of the project is available and ready to be used : Amazon Web Services for hosting the database.
   - All members of the team are sufficiently skilled to see the project through to its completion
3. Schedule:
   - The project is estimated to be finished before the given deadline.
   - The team's focus is to create the minimum viable product with the necessary functionality in order to ensure a product can be delivered before the deadline.
   - There will be daily meetings to assign tasks and check progress on already existing tasks.
4. *Legal:*
   - GDPR - storing of the people's data such as birthdates, etc. Therefore, we offer an ability for Parents to request information on what data we store about them as well as complete deletion from the system.
     All users have to give consent for storing the personal information when they join our Daycare system.

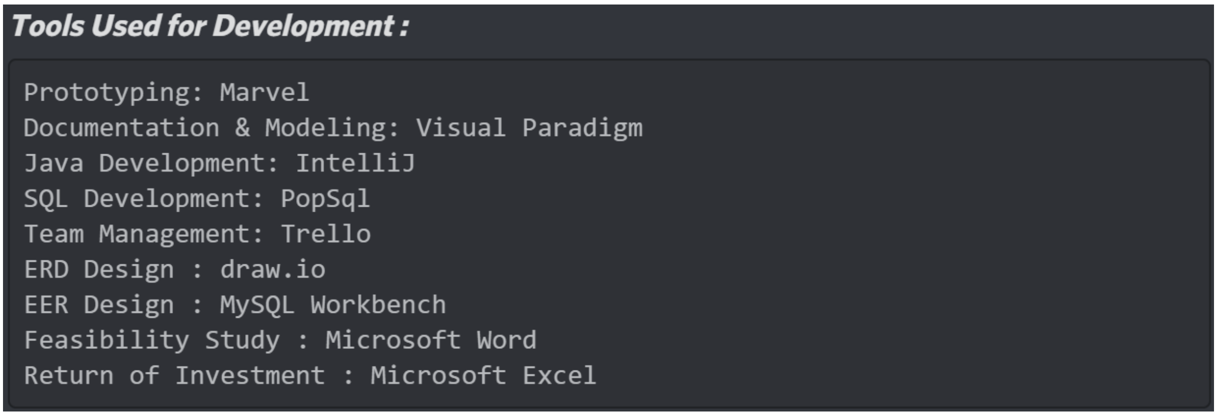
5. *Political Feasibility:*
   - The key stakeholders : Marianne, Douglas & Cay approve of the project and support it.
6. *Economic Feasibility:*

RoskildeDaycare
RoI.xlsx

Description Of Development

The development of this project was done in iterations, following the UP model (to the best of our abilities), with the help of the following tools :

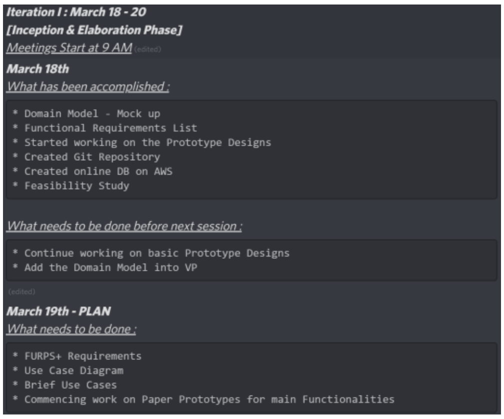
Discord – Managing the team and the progress + Arranging meetings.



Tools Used for Development :

```
Prototyping: Marvel
Documentation & Modeling: Visual Paradigm
Java Development: IntelliJ
SQL Development: PopSql
Team Management: Trello
ERD Design : draw.io
EER Design : MySQL Workbench
Feasibility Study : Microsoft Word
Return of Investment : Microsoft Excel
```

For keeping track of what needs to be done each day + what has been covered each day, as well as the current progress within each iteration, we used a channel in Discord, where an agenda was maintained (more or less daily) :



Iteration I : March 18 - 20
[Inception & Elaboration Phase]
Meetings Start at 9 AM (edited)

March 18th
What has been accomplished :

```
* Domain Model - Mock up
* Functional Requirements List
* Started working on the Prototype Designs
* Created Git Repository
* Created online DB on AWS
* Feasibility Study
```

What needs to be done before next session :

```
* Continue working on basic Prototype Designs
* Add the Domain Model into VP
```

(edited)

March 19th - PLAN
What needs to be done :

```
* FURPS+ Requirements
* Use Case Diagram
* Brief Use Cases
* Commencing work on Paper Prototypes for main Functionalities
```

The Development started on March 18, with the first Iteration, planned to cover the Inception Phase. However, due to the small scale of the project, we ended up covering the Elaboration Phase as well.

We determined the requirements, done the feasibility study to determine whether the project was worth completing (we would have completed it anyways) and started creating diagrams, such as the Domain Model & Use Case Diagram.

Paper Prototypes have been done in order to establish how the flow of the UI should be like, as well as what other requirements and functionalities need to be added.

At the end of this Iteration, we also created the first version of the Class Diagram (which had to be completely redone once we discovered how big the impact of interacting with a database was on the structure of the code).

The following Iteration (Construction Phase) started on 21st of March and was devoted solely to coding the MVP (Minimum Viable Product).

One part of the team focused on the Database creation & population (with realistic data & information);
In the meantime, the other part of the team focused on creating the structure of the Java code in IntelliJ and preparing for writing the actual functioning queries on the DB.

The next step was to update the Class Diagram to better reflect the new Structure of the program we had in mind.

After all of us understood how the connection to the Database is being done, as well as how to write actual queries that affect & retrieve information live from the DB, we started defining what the queries should look like so that we would achieve the desired result & we coded them, bringing this Iteration to an end.

The final Iteration, which comes with the Deployment Phase, started on the 24th of March and mainly consisted of bug-fixing, updating the CLI & overall making sure that the program works properly.

All Diagrams can be seen in the Appendices Section.

The Database

In order to create the Database, we first designed a quick ERD (Entity Relationship Diagram) in order to wrap our heads around how the DB should look like.

Later, based on the ERD we later created an EERD (Enhanced Entity Relationship Diagram) in MySQL Workbench to properly showcase all entities and the relationships between them.

The DB creation code was then generated by MySQL Workbench (based on the EERD) and then had to be edited to better suit our needs.

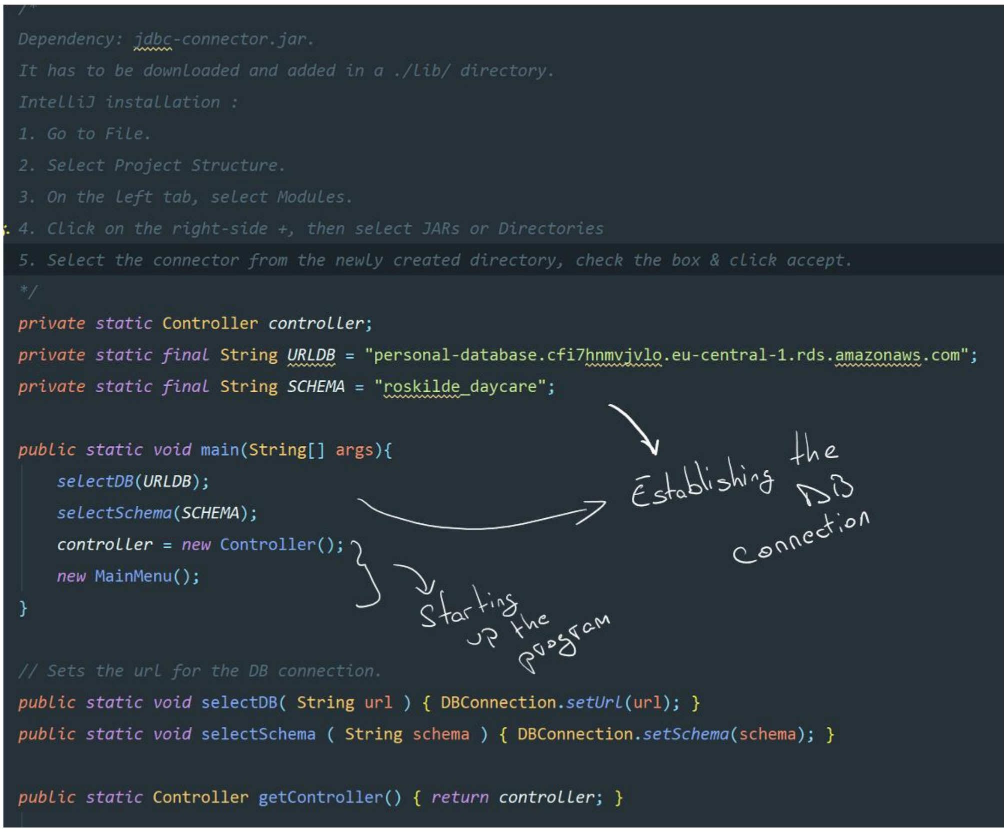
After creating the DB, we worked on populating it with realistic data, which helped in developing a proper vision on how the data should be used.

After we had the database all ready to be used, we created a User Management System so that the DB can properly work with the Login System created in the Java Program (we have created an admin user & users for all teacher (also included user creation for every time a teacher is created in the database) all with proper, secure, passwords seeing as the database can pretty much be accessed by anyone with the right credentials).

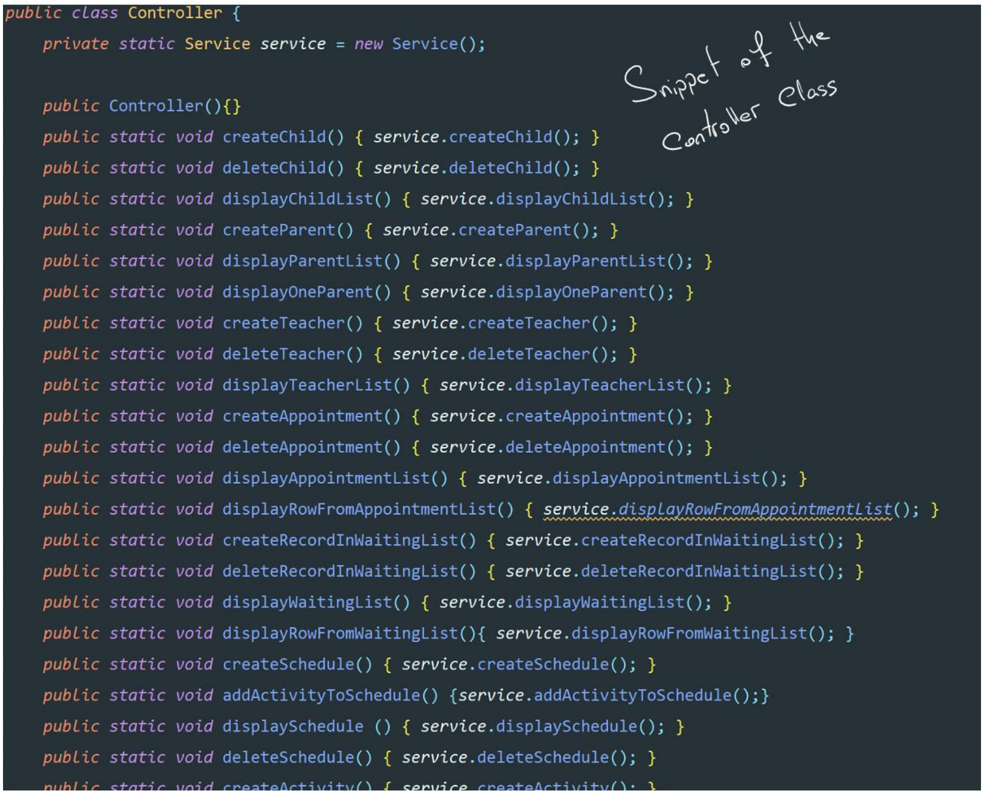
The diagrams & snippets of SQL code can be found in Appendices Section, or here.

Java Development

In our program we tried to make our information flow as clear as we could. We designed class menu to display available features to user and send information about needed information to controller. Next, controller class sends information to service class in which we extract information from database and display what needed to the user.

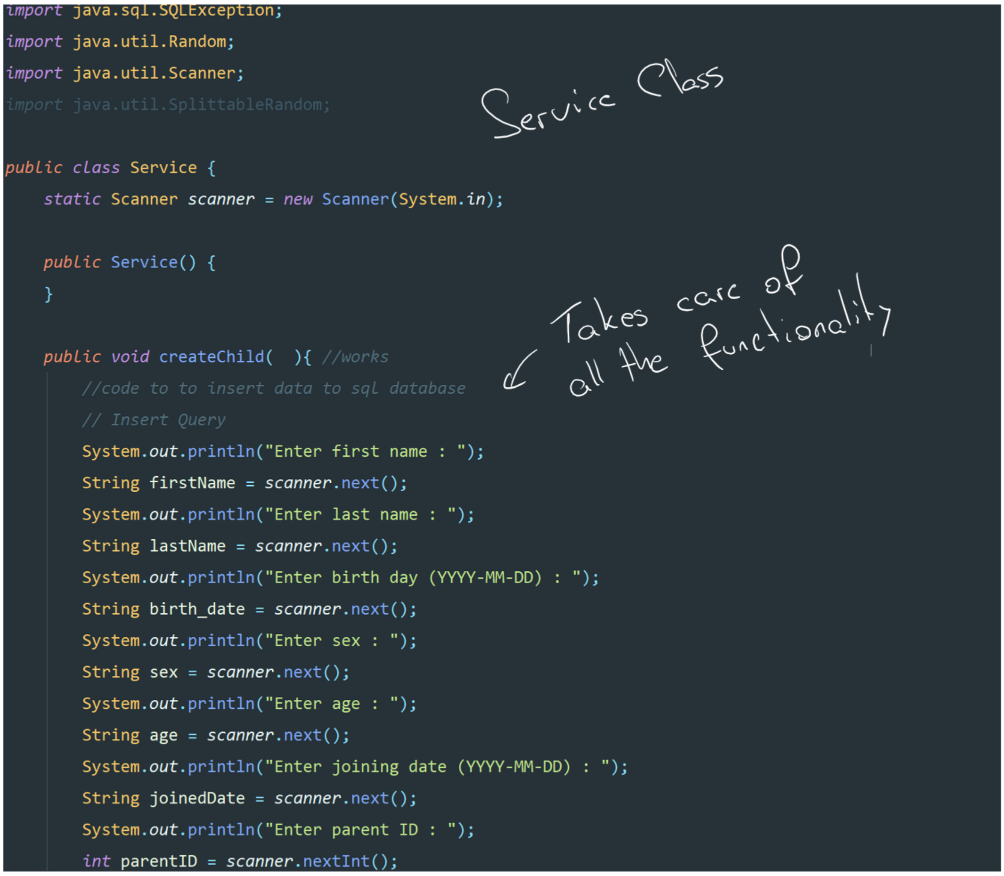


```
/*
Dependency: jdbc-connector.jar.
It has to be downloaded and added in a ./lib/ directory.
IntelliJ installation :
1. Go to File.
2. Select Project Structure.
3. On the left tab, select Modules.
4. Click on the right-side +, then select JARs or Directories
5. Select the connector from the newly created directory, check the box & click accept.
*/
private static Controller controller;
private static final String URLDB = "personal-database.cfi7hnmvjvlo.eu-central-1.rds.amazonaws.com";
private static final String SCHEMA = "roskilde_daycare";

public static void main(String[] args){
    selectDB(URLDB);
    selectSchema(SCHEMA);
    controller = new Controller();
    new MainMenu();
}

// Sets the url for the DB connection.
public static void selectDB( String url ) { DBConnection.setUrl(url); }
public static void selectSchema ( String schema ) { DBConnection.setSchema(schema); }

public static Controller getController() { return controller; }
```



```java
public class Controller {
    private static Service service = new Service();

    public Controller(){}
    public static void createChild() { service.createChild(); }
    public static void deleteChild() { service.deleteChild(); }
    public static void displayChildList() { service.displayChildList(); }
    public static void createParent() { service.createParent(); }
    public static void displayParentList() { service.displayParentList(); }
    public static void displayOneParent() { service.displayOneParent(); }
    public static void createTeacher() { service.createTeacher(); }
    public static void deleteTeacher() { service.deleteTeacher(); }
    public static void displayTeacherList() { service.displayTeacherList(); }
    public static void createAppointment() { service.createAppointment(); }
    public static void deleteAppointment() { service.deleteAppointment(); }
    public static void displayAppointmentList() { service.displayAppointmentList(); }
    public static void displayRowFromAppointmentList() { service.displayRowFromAppointmentList(); }
    public static void createRecordInWaitingList() { service.createRecordInWaitingList(); }
    public static void deleteRecordInWaitingList() { service.deleteRecordInWaitingList(); }
    public static void displayWaitingList() { service.displayWaitingList(); }
    public static void displayRowFromWaitingList(){ service.displayRowFromWaitingList(); }
    public static void createSchedule() { service.createSchedule(); }
    public static void addActivityToSchedule() {service.addActivityToSchedule();}
    public static void displaySchedule () { service.displaySchedule(); }
    public static void deleteSchedule() { service.deleteSchedule(); }
    public static void createActivity() { service.createActivity(); }
```

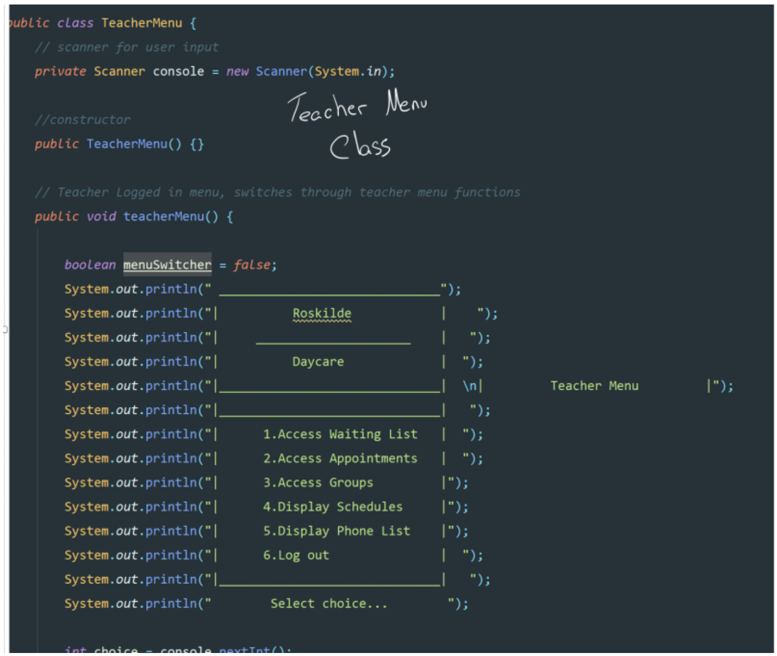
Snippet of the Controller Class



```java
import java.sql.SQLException;
import java.util.Random;
import java.util.Scanner;
import java.util.SplittableRandom;

public class Service {
    static Scanner scanner = new Scanner(System.in);

    public Service() {
    }

    public void createChild(   ){ //works
        //code to to insert data to sql database
        // Insert Query
        System.out.println("Enter first name : ");
        String firstName = scanner.next();
        System.out.println("Enter last name : ");
        String lastName = scanner.next();
        System.out.println("Enter birth day (YYYY-MM-DD) : ");
        String birth_date = scanner.next();
        System.out.println("Enter sex : ");
        String sex = scanner.next();
        System.out.println("Enter age : ");
        String age = scanner.next();
        System.out.println("Enter joining date (YYYY-MM-DD) : ");
        String joinedDate = scanner.next();
        System.out.println("Enter parent ID : ");
        int parentID = scanner.nextInt();
```

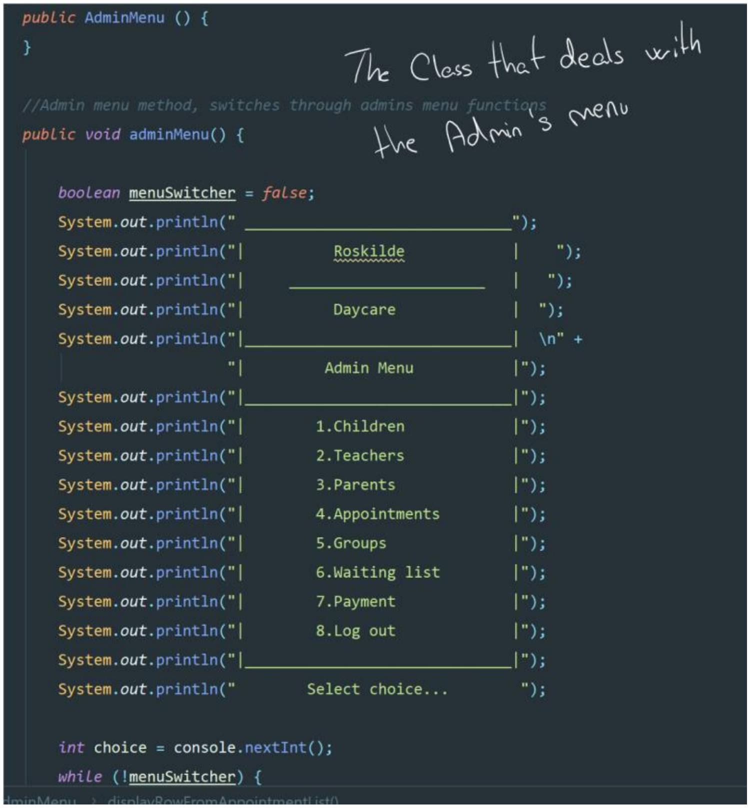
Service Class

Takes care of all the functionality



```java
public class TeacherMenu {
    // scanner for user input
    private Scanner console = new Scanner(System.in);

    //constructor
    public TeacherMenu() {}

    // Teacher Logged in menu, switches through teacher menu functions
    public void teacherMenu() {

        boolean menuSwitcher = false;
        System.out.println(" ______________________________");
        System.out.println("|          Roskilde            |    ");
        System.out.println("|     _____________________    |   ");
        System.out.println("|          Daycare             |  ");
        System.out.println("|______________________________|  \n|         Teacher Menu         |");
        System.out.println("|______________________________|   ");
        System.out.println("|      1.Access Waiting List   |  ");
        System.out.println("|      2.Access Appointments   |  ");
        System.out.println("|      3.Access Groups         |");
        System.out.println("|      4.Display Schedules     |");
        System.out.println("|      5.Display Phone List    |");
        System.out.println("|      6.Log out               |  ");
        System.out.println("|______________________________|   ");
        System.out.println("       Select choice...        ");

        int choice = console.nextInt();
```

Teacher Menu
Class



```java
public AdminMenu () {
}

//Admin menu method, switches through admins menu functions
public void adminMenu() {

    boolean menuSwitcher = false;
    System.out.println(" ______________________________");
    System.out.println("|          Roskilde            |    ");
    System.out.println("|     ______________________   |   ");
    System.out.println("|          Daycare             |  ");
    System.out.println("|______________________________|  \n" +
                   "|          Admin Menu          |");
    System.out.println("|______________________________|");
    System.out.println("|        1.Children            |");
    System.out.println("|        2.Teachers            |");
    System.out.println("|        3.Parents             |");
    System.out.println("|        4.Appointments        |");
    System.out.println("|        5.Groups              |");
    System.out.println("|        6.Waiting list        |");
    System.out.println("|        7.Payment             |");
    System.out.println("|        8.Log out             |");
    System.out.println("|______________________________|");
    System.out.println("        Select choice...       ");

    int choice = console.nextInt();
    while (!menuSwitcher) {
```

dminMenu > displayRowFromAppointmentList()

The Class that deals with
the Admin's menu

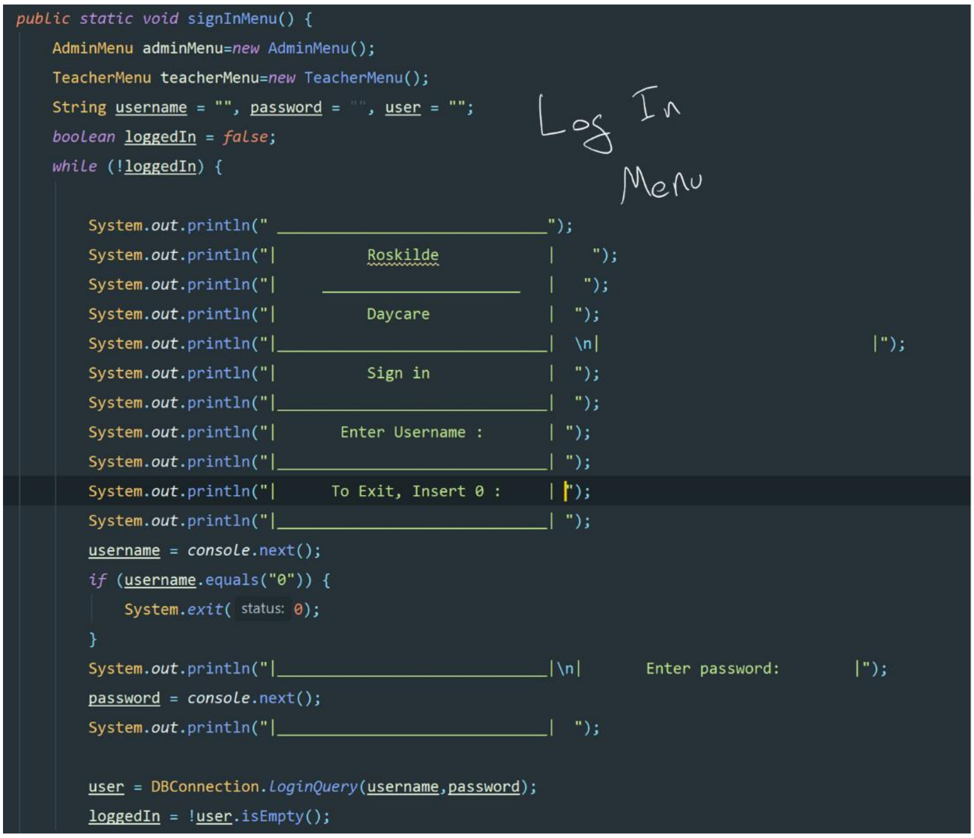
Snippets Of Code

We have decided to present snippets of code :

- Log in as administrator
- Create teacher
- Delete child
- Display list of parents

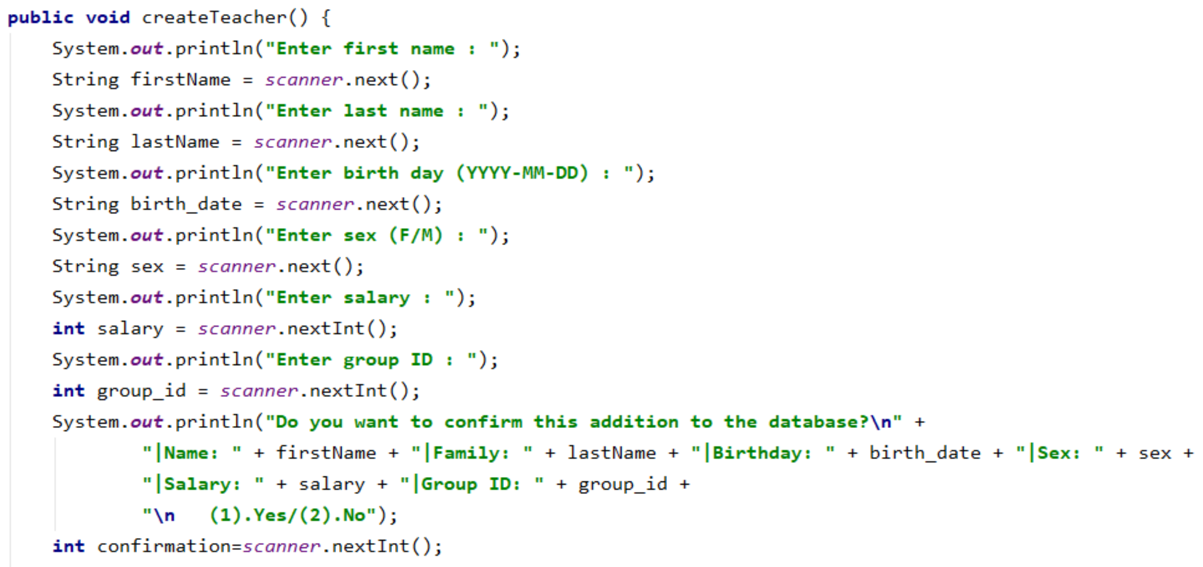
These methods represent general idea how our program works. In order to see the rest of the methods we encourage you to look at our source code attached to the report.

Log in as administrator

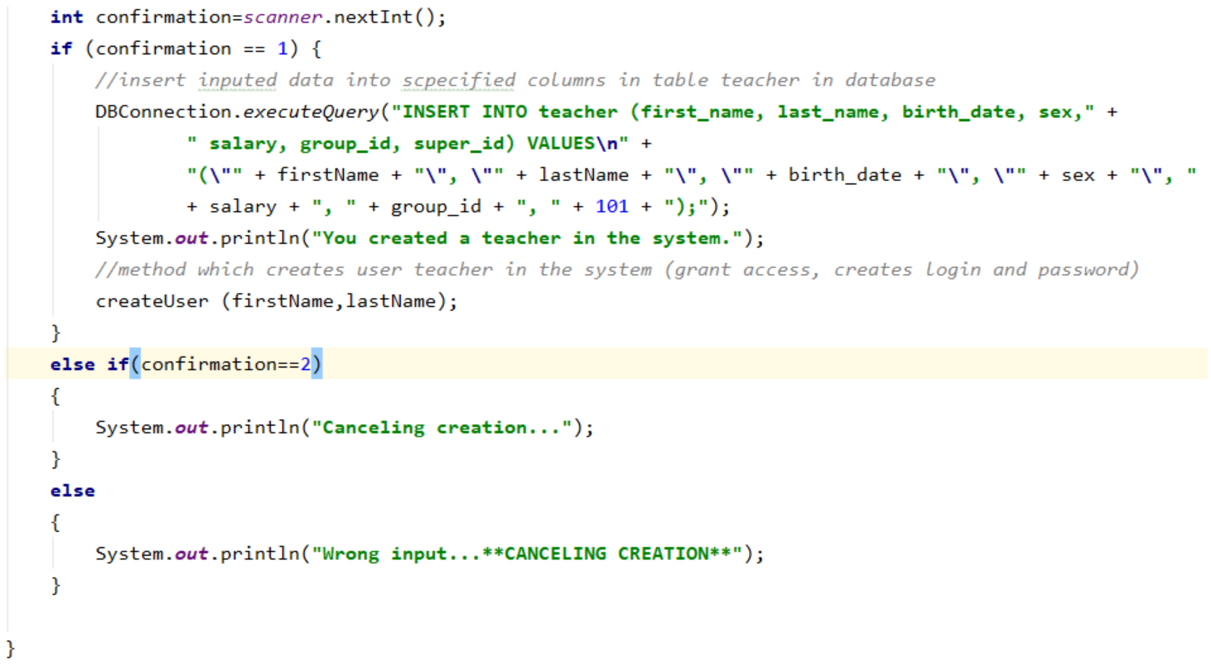


```
public static void signInMenu() {
    AdminMenu adminMenu=new AdminMenu();
    TeacherMenu teacherMenu=new TeacherMenu();
    String username = "", password = "", user = "";
    boolean loggedIn = false;
    while (!loggedIn) {

        System.out.println(" ______________________________");
        System.out.println("|          Roskilde            |    ");
        System.out.println("|     ______________________   |   ");
        System.out.println("|          Daycare             |  ");
        System.out.println("|______________________________|  \n|                             |");
        System.out.println("|          Sign in             |  ");
        System.out.println("|______________________________|  ");
        System.out.println("|       Enter Username :       | ");
        System.out.println("|______________________________| ");
        System.out.println("|      To Exit, Insert 0 :     | ");
        System.out.println("|______________________________| ");
        username = console.next();
        if (username.equals("0")) {
            System.exit( status: 0);
        }
        System.out.println("|______________________________|\n|       Enter password:       |");
        password = console.next();
        System.out.println("|______________________________|  ");

        user = DBConnection.loginQuery(username,password);
        loggedIn = !user.isEmpty();
```

Create teacher method

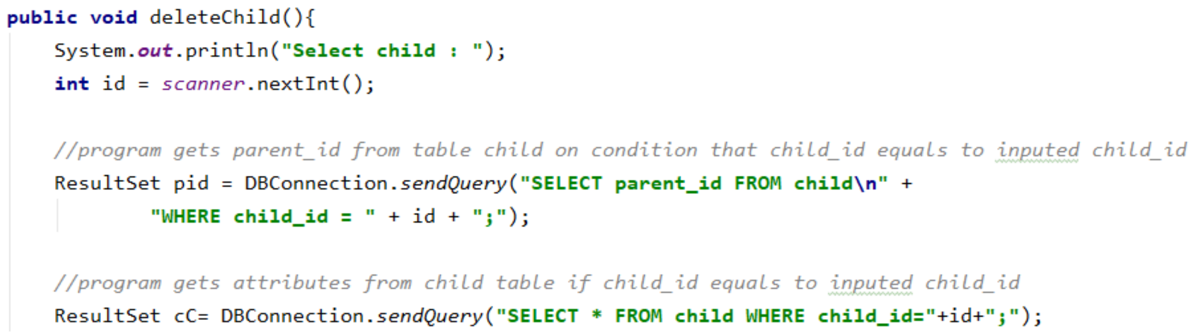


```java
public void createTeacher() {
    System.out.println("Enter first name : ");
    String firstName = scanner.next();
    System.out.println("Enter last name : ");
    String lastName = scanner.next();
    System.out.println("Enter birth day (YYYY-MM-DD) : ");
    String birth_date = scanner.next();
    System.out.println("Enter sex (F/M) : ");
    String sex = scanner.next();
    System.out.println("Enter salary : ");
    int salary = scanner.nextInt();
    System.out.println("Enter group ID : ");
    int group_id = scanner.nextInt();
    System.out.println("Do you want to confirm this addition to the database?\n" +
            "|Name: " + firstName + "|Family: " + lastName + "|Birthday: " + birth_date + "|Sex: " + sex +
            "|Salary: " + salary + "|Group ID: " + group_id +
            "\n   (1).Yes/(2).No");
    int confirmation=scanner.nextInt();
```

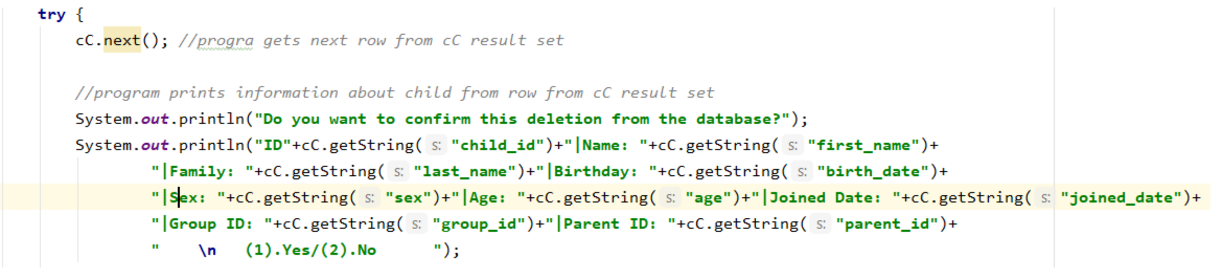


```java
    int confirmation=scanner.nextInt();
    if (confirmation == 1) {
        //insert inputed data into scpecified columns in table teacher in database
        DBConnection.executeQuery("INSERT INTO teacher (first_name, last_name, birth_date, sex," +
                " salary, group_id, super_id) VALUES\n" +
                "(\"" + firstName + "\", \"" + lastName + "\", \"" + birth_date + "\", \"" + sex + "\", "
                + salary + ", " + group_id + ", " + 101 + ");");
        System.out.println("You created a teacher in the system.");
        //method which creates user teacher in the system (grant access, creates login and password)
        createUser (firstName,lastName);
    }
    else if(confirmation==2)
    {
        System.out.println("Canceling creation...");
    }
    else
    {
        System.out.println("Wrong input...**CANCELING CREATION**");
    }

}
```

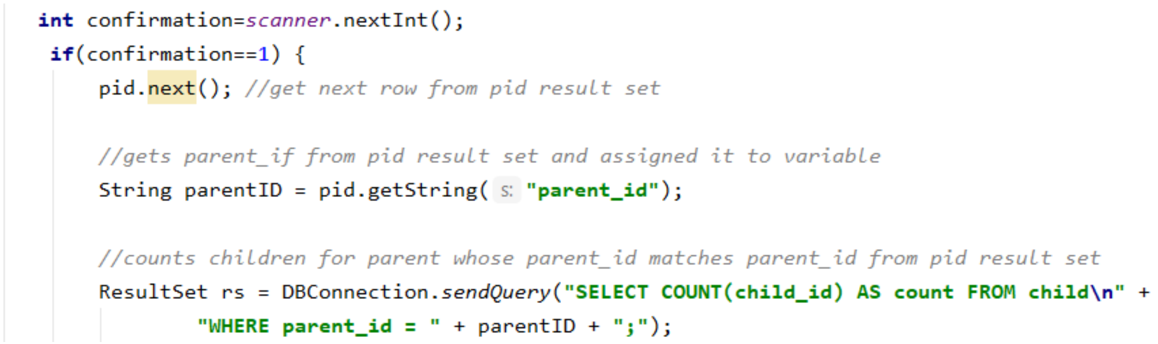
Delete child



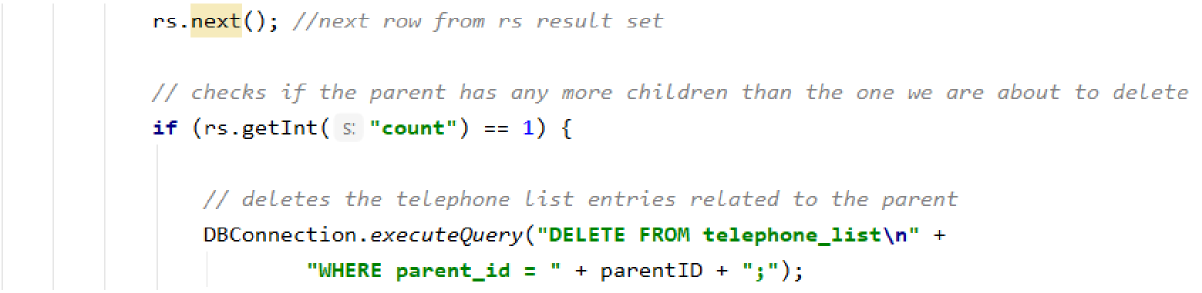
```java
public void deleteChild(){
    System.out.println("Select child : ");
    int id = scanner.nextInt();

    //program gets parent_id from table child on condition that child_id equals to inputed child_id
    ResultSet pid = DBConnection.sendQuery("SELECT parent_id FROM child\n" +
            "WHERE child_id = " + id + ";");

    //program gets attributes from child table if child_id equals to inputed child_id
    ResultSet cC= DBConnection.sendQuery("SELECT * FROM child WHERE child_id="+id+";");
```



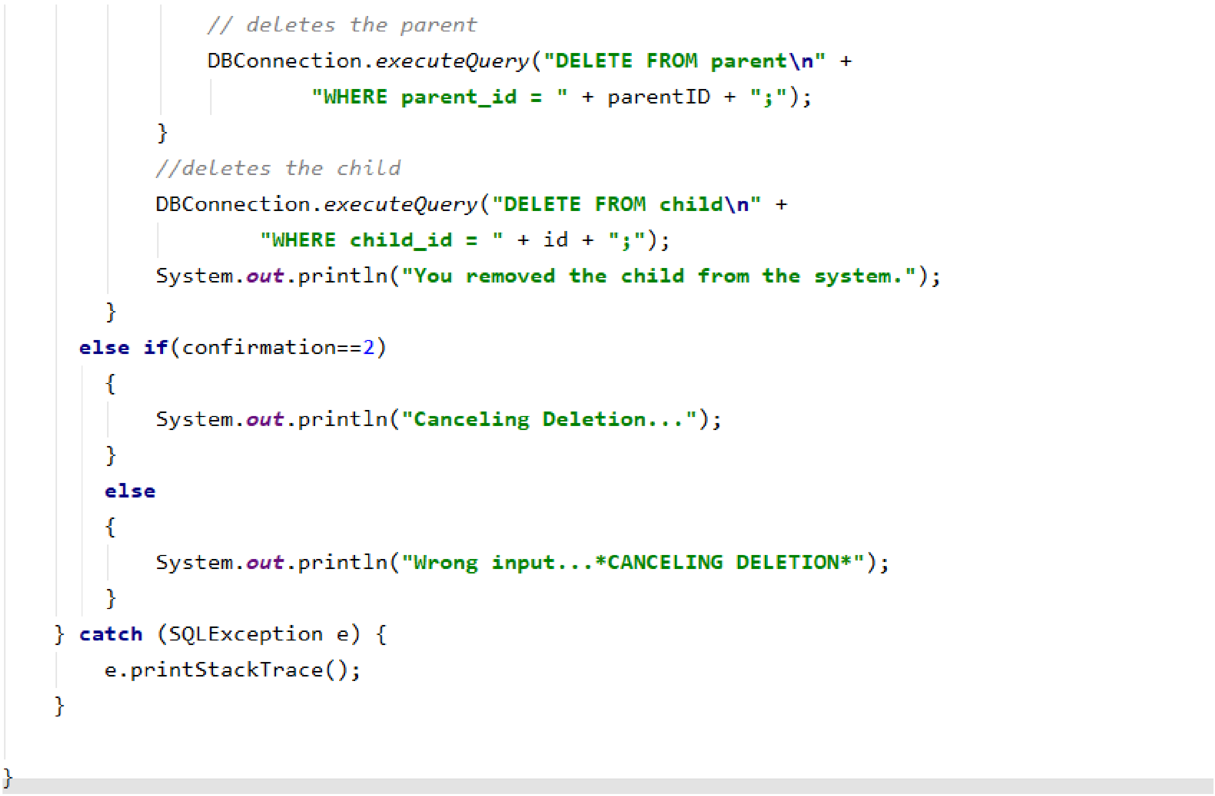
```
try {
    cC.next(); //progra gets next row from cC result set

    //program prints information about child from row from cC result set
    System.out.println("Do you want to confirm this deletion from the database?");
    System.out.println("ID"+cC.getString( s: "child_id")+"|Name: "+cC.getString( s: "first_name")+
            "|Family: "+cC.getString( s: "last_name")+"|Birthday: "+cC.getString( s: "birth_date")+
            "|Sex: "+cC.getString( s: "sex")+"|Age: "+cC.getString( s: "age")+"|Joined Date: "+cC.getString( s: "joined_date")+
            "|Group ID: "+cC.getString( s: "group_id")+"|Parent ID: "+cC.getString( s: "parent_id")+
            "    \n   (1).Yes/(2).No      ");
```



```
 int confirmation=scanner.nextInt();
  if(confirmation==1) {
      pid.next(); //get next row from pid result set

      //gets parent_if from pid result set and assigned it to variable
      String parentID = pid.getString( s: "parent_id");

      //counts children for parent whose parent_id matches parent_id from pid result set
      ResultSet rs = DBConnection.sendQuery("SELECT COUNT(child_id) AS count FROM child\n" +
              "WHERE parent_id = " + parentID + ";");
```

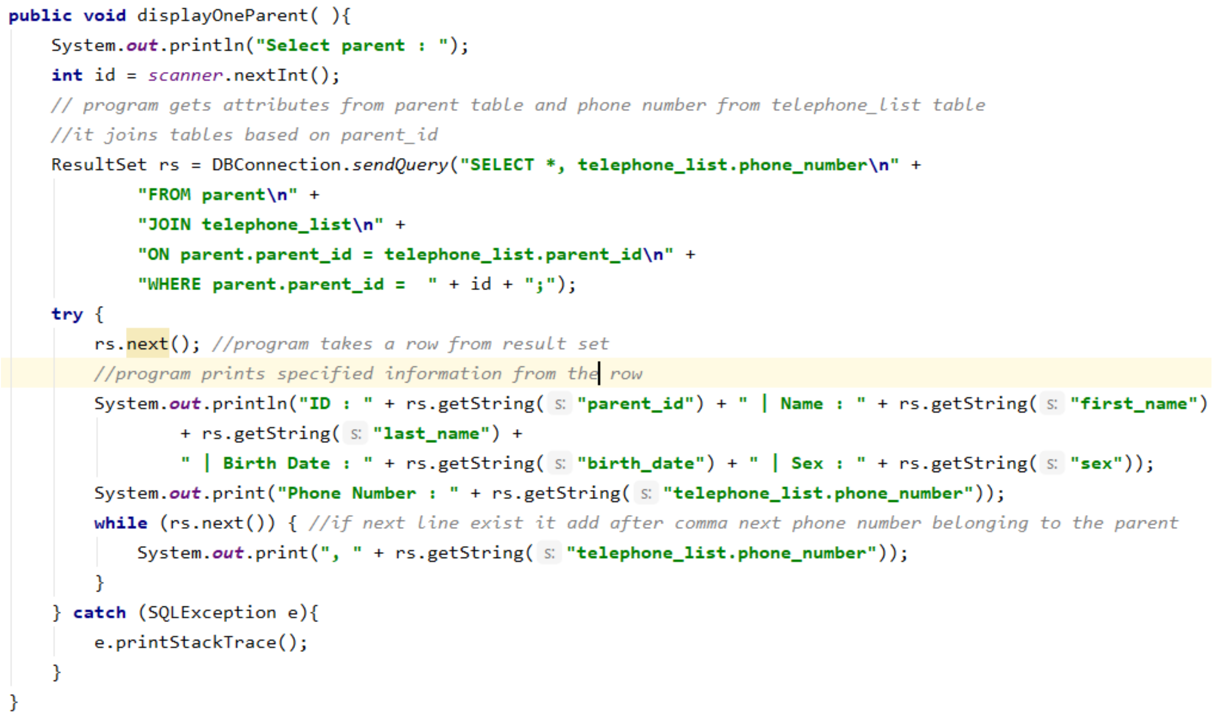


```
       rs.next(); //next row from rs result set

       // checks if the parent has any more children than the one we are about to delete
       if (rs.getInt( s: "count") == 1) {

           // deletes the telephone list entries related to the parent
           DBConnection.executeQuery("DELETE FROM telephone_list\n" +
                   "WHERE parent_id = " + parentID + ";");
```



```
           // deletes the parent
           DBConnection.executeQuery("DELETE FROM parent\n" +
                   "WHERE parent_id = " + parentID + ";");
       }
       //deletes the child
       DBConnection.executeQuery("DELETE FROM child\n" +
               "WHERE child_id = " + id + ";");
       System.out.println("You removed the child from the system.");
   }
  else if(confirmation==2)
   {
       System.out.println("Canceling Deletion...");
   }
   else
   {
       System.out.println("Wrong input...*CANCELING DELETION*");
   }
} catch (SQLException e) {
    e.printStackTrace();
}

}
```

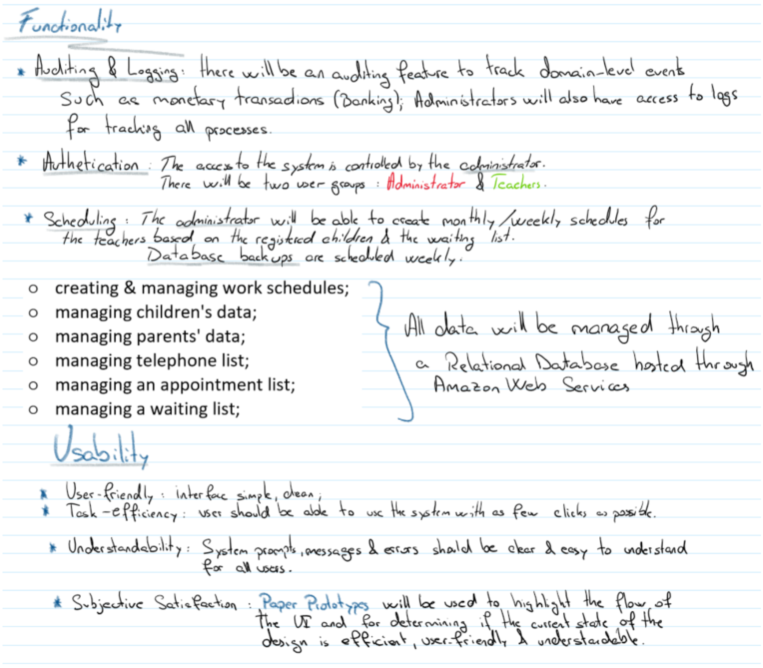
Display list of parents



```
public void displayOneParent( ){
    System.out.println("Select parent : ");
    int id = scanner.nextInt();
    // program gets attributes from parent table and phone number from telephone_list table
    //it joins tables based on parent_id
    ResultSet rs = DBConnection.sendQuery("SELECT *, telephone_list.phone_number\n" +
            "FROM parent\n" +
            "JOIN telephone_list\n" +
            "ON parent.parent_id = telephone_list.parent_id\n" +
            "WHERE parent.parent_id =  " + id + ";");
    try {
        rs.next(); //program takes a row from result set
        //program prints specified information from the row
        System.out.println("ID : " + rs.getString( s: "parent_id") + " | Name : " + rs.getString( s: "first_name")
                + rs.getString( s: "last_name") +
                " | Birth Date : " + rs.getString( s: "birth_date") + " | Sex : " + rs.getString( s: "sex"));
        System.out.print("Phone Number : " + rs.getString( s: "telephone_list.phone_number"));
        while (rs.next()) { //if next line exist it add after comma next phone number belonging to the parent
            System.out.print(", " + rs.getString( s: "telephone_list.phone_number"));
        }
    } catch (SQLException e){
        e.printStackTrace();
    }
}
```

Appendices

FURPS+



Functionality

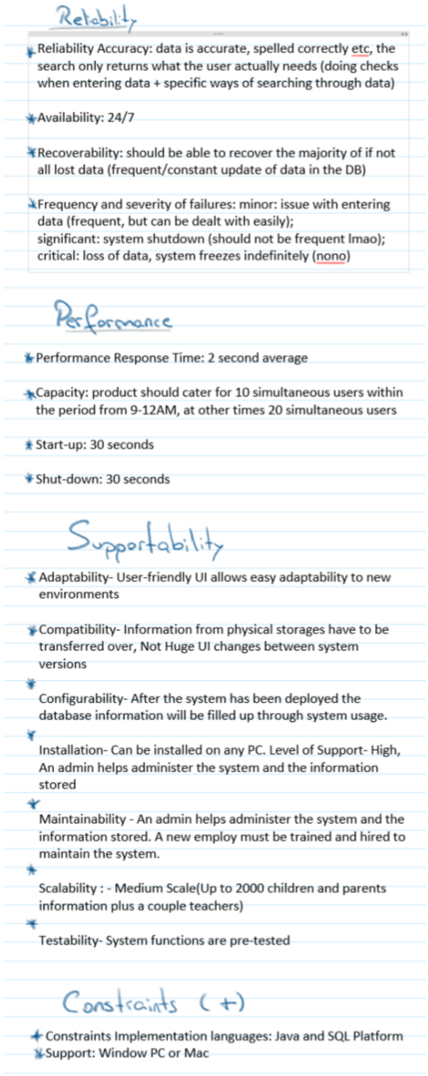
- Auditing & Logging: there will be an auditing feature to track domain-level events Such as monetary transactions (Banking); Administrators will also have access to logs for tracking all processes.
- Authetication : The access to the system is controlled by the administrator. There will be two user groups : Administrator & Teachers.
- Scheduling : The administrator will be able to create monthly /weekly schedules for the teachers based on the registered children & the waiting list. Database backups are scheduled weekly.

- creating & managing work schedules;
- managing children's data;
- managing parents' data;
- managing telephone list;
- managing an appointment list;
- managing a waiting list;

All data will be managed through a Relational Database hosted through Amazon Web Services

Usability

- User-friendly : interface simple, clean;
- Task-efficiency : user should be able to use the system with as few clicks as possible.
- Understandability : System prompts, messages & errors should be clear & easy to understand for all users.
- Subjective Satisfaction : Paper Prototypes will be used to highlight the flow of the UI and for determining if the current state of the design is efficient, user-friendly & understandable.



Reliability

- Reliability Accuracy: data is accurate, spelled correctly etc, the search only returns what the user actually needs (doing checks when entering data + specific ways of searching through data)
- Availability: 24/7
- Recoverability: should be able to recover the majority of if not all lost data (frequent/constant update of data in the DB)
- Frequency and severity of failures: minor: issue with entering data (frequent, but can be dealt with easily); significant: system shutdown (should not be frequent lmao); critical: loss of data, system freezes indefinitely (nono)

Performance

- Performance Response Time: 2 second average
- Capacity: product should cater for 10 simultaneous users within the period from 9-12AM, at other times 20 simultaneous users
- Start-up: 30 seconds
- Shut-down: 30 seconds

Supportability

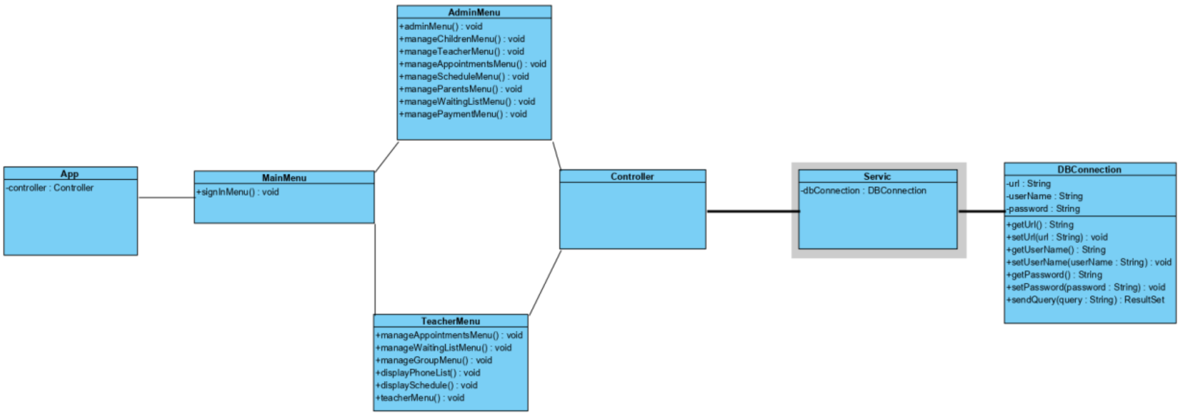
- Adaptability- User-friendly UI allows easy adaptability to new environments
- Compatibility- Information from physical storages have to be transferred over, Not Huge UI changes between system versions
- Configurability- After the system has been deployed the database information will be filled up through system usage.
- Installation- Can be installed on any PC. Level of Support- High, An admin helps administer the system and the information stored
- Maintainability - An admin helps administer the system and the information stored. A new employ must be trained and hired to maintain the system.
- Scalability : - Medium Scale(Up to 2000 children and parents information plus a couple teachers)
- Testability- System functions are pre-tested

Constraints (+)

- Constraints Implementation languages: Java and SQL Platform
- Support: Window PC or Mac

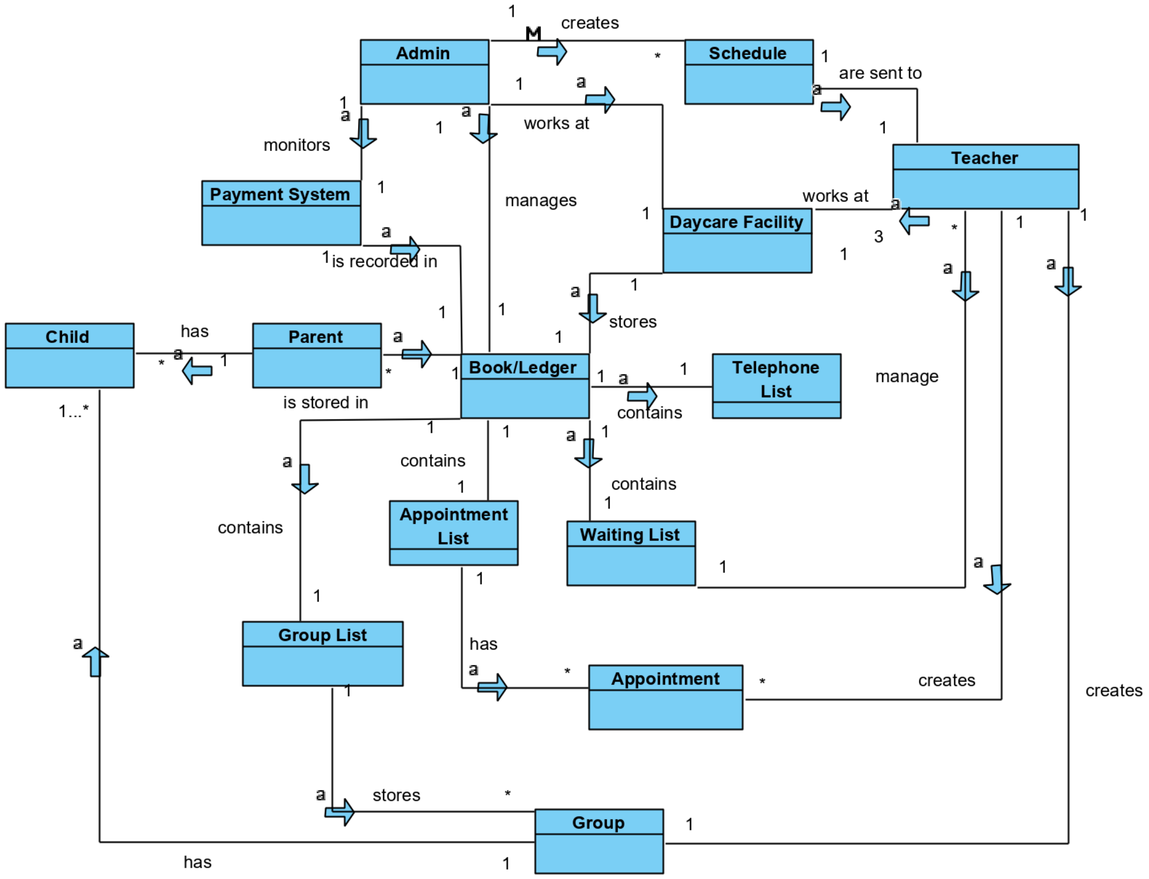
Diagrams

Class Diagram

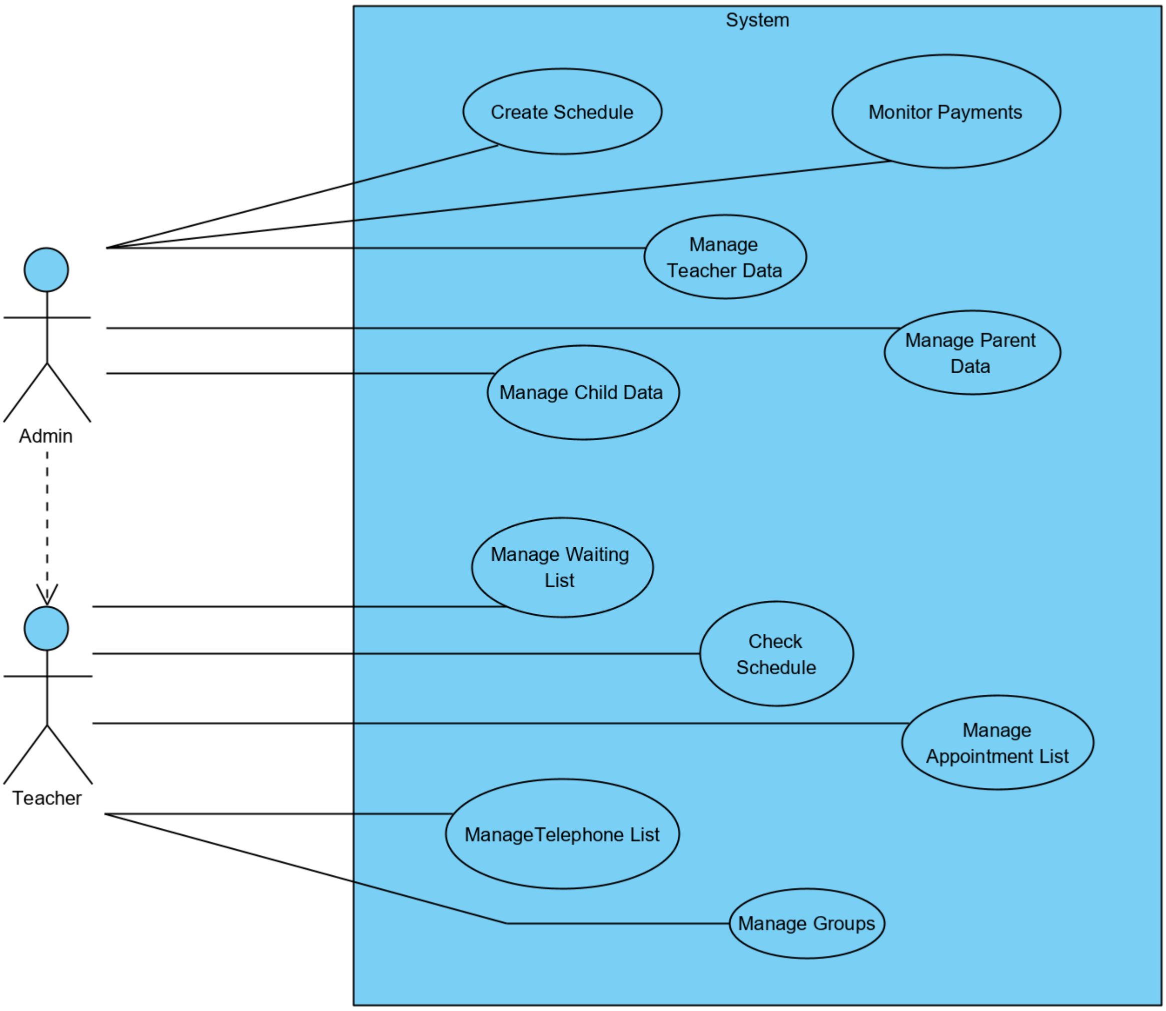


App
-controller : Controller
MainMenu
+signInMenu() : void
AdminMenu
+adminMenu() : void
+manageChildrenMenu() : void
+manageTeacherMenu() : void
+manageAppointmentsMenu() : void
+manageScheduleMenu() : void
+manageParentsMenu() : void
+manageWaitingListMenu() : void
+managePaymentMenu() : void
TeacherMenu
+manageAppointmentsMenu() : void
+manageWaitingListMenu() : void
+manageGroupMenu() : void
+displayPhoneList() : void
+displaySchedule() : void
+teacherMenu() : void
Controller
Servic
-dbConnection : DBConnection
DBConnection
-url : String
-userName : String
-password : String
+getUrl() : String
+setUrl(url : String) : void
+getUserName() : String
+setUserName(userName : String) : void
+getPassword() : String
+setPassword(password : String) : void
+sendQuery(query : String) : ResultSet

Domain Model

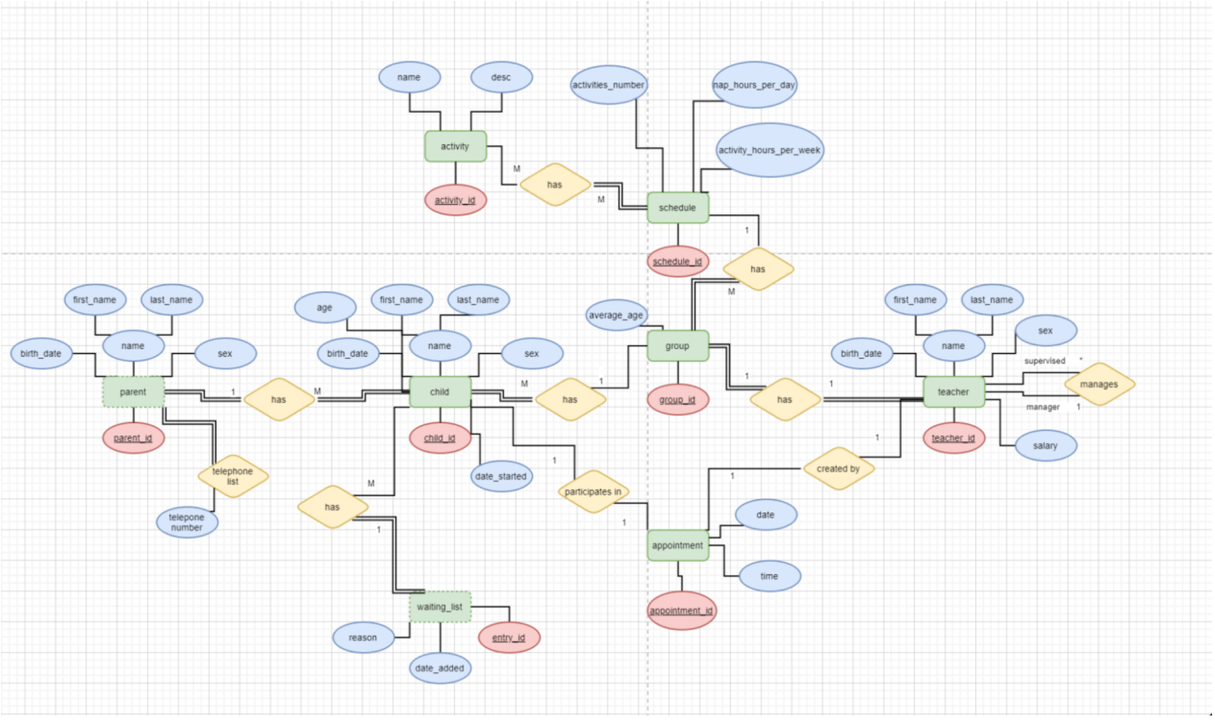


Use-Case Diagram

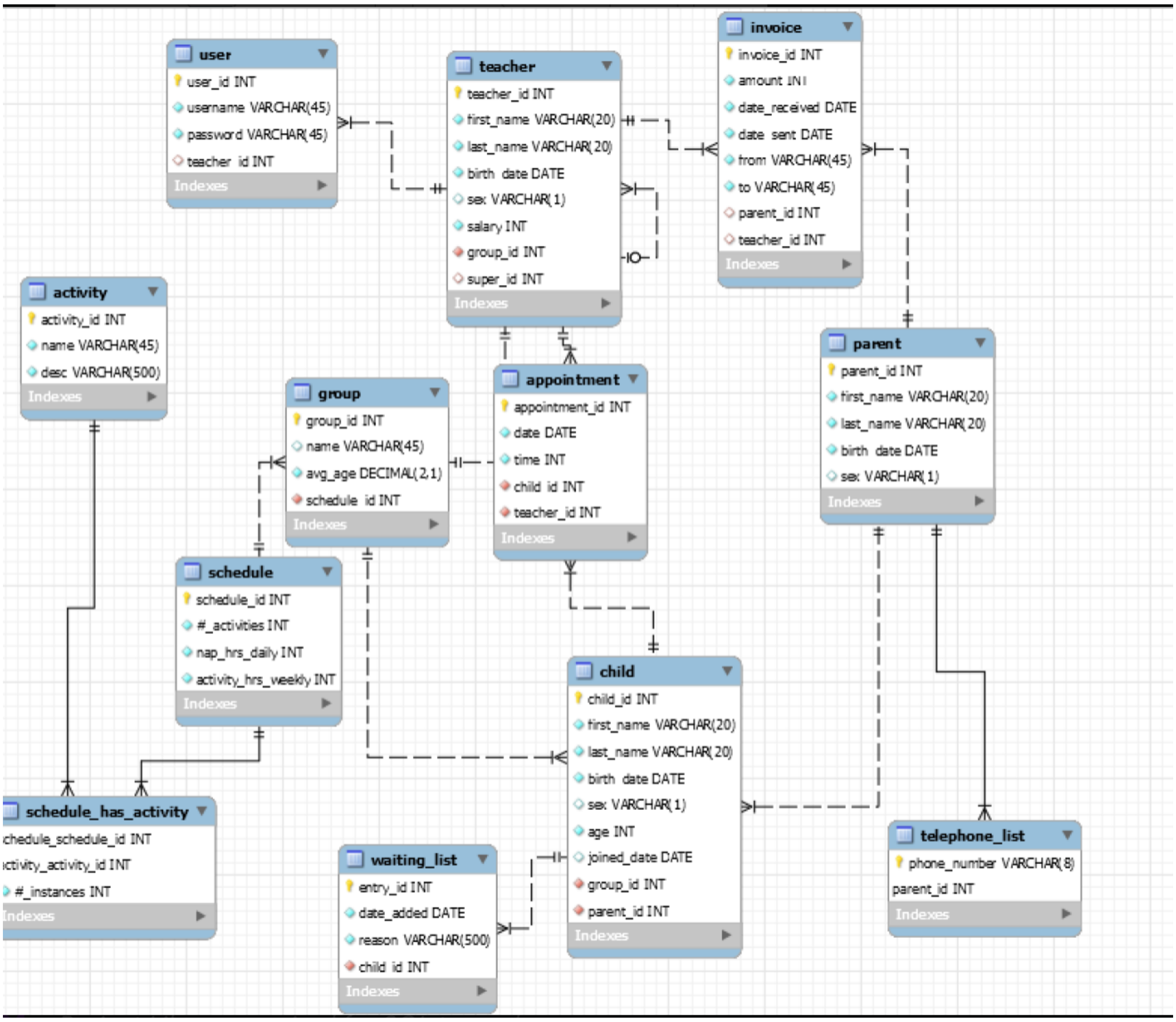


Database

ERD

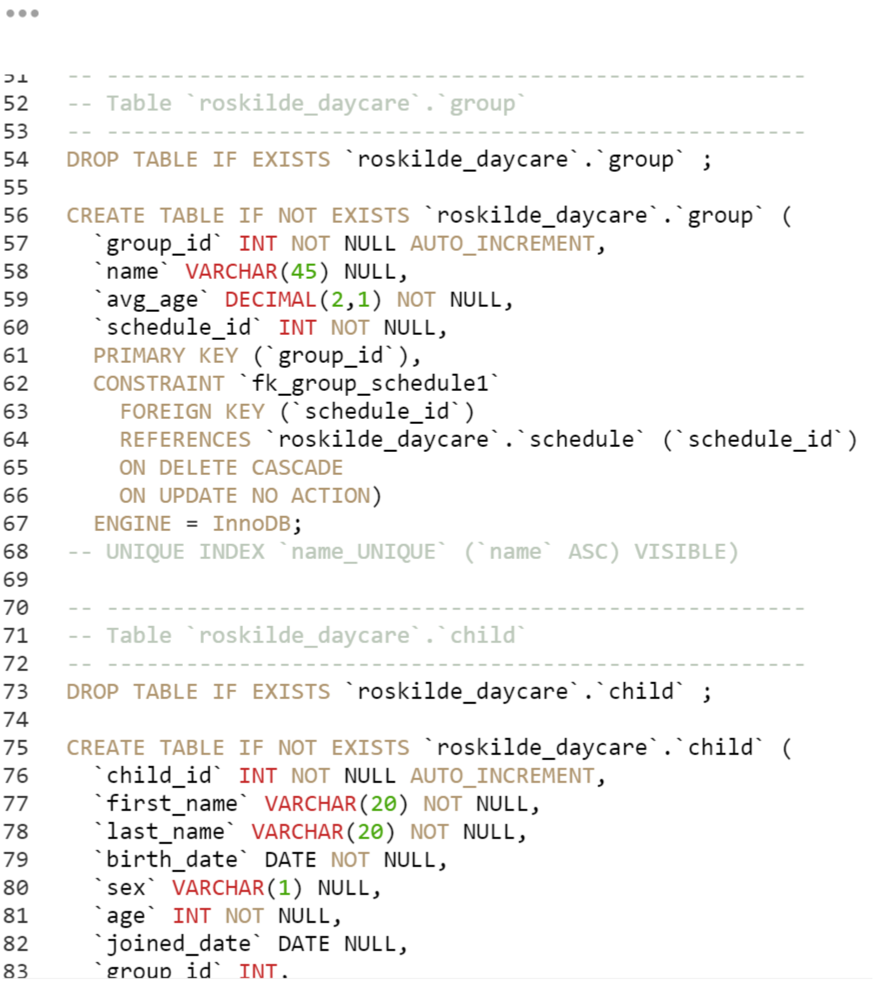


EERD


user
user_id INT
username VARCHAR(45)
password VARCHAR(45)
teacher_id INT
Indexes
teacher
teacher_id INT
first_name VARCHAR(20)
last_name VARCHAR(20)
birth_date DATE
sex VARCHAR(1)
salary INT
group_id INT
super_id INT
Indexes
invoice
invoice_id INT
amount INT
date_received DATE
date_sent DATE
from VARCHAR(45)
to VARCHAR(45)
parent_id INT
teacher_id INT
Indexes
activity
activity_id INT
name VARCHAR(45)
desc VARCHAR(500)
Indexes
group
group_id INT
name VARCHAR(45)
avg_age DECIMAL(2,1)
schedule_id INT
Indexes
appointment
appointment_id INT
date DATE
time INT
child_id INT
teacher_id INT
Indexes
parent
parent_id INT
first_name VARCHAR(20)
last_name VARCHAR(20)
birth_date DATE
sex VARCHAR(1)
Indexes
schedule
schedule_id INT
#_activities INT
nap_hrs_daily INT
activity_hrs_weekly INT
Indexes
child
child_id INT
first_name VARCHAR(20)
last_name VARCHAR(20)
birth_date DATE
sex VARCHAR(1)
age INT
joined_date DATE
group_id INT
parent_id INT
Indexes
schedule_has_activity
chedule_schedule_id INT
ctivity_activity_id INT
#_instances INT
ndexes
waiting_list
entry_id INT
date_added DATE
reason VARCHAR(500)
child_id INT
Indexes
telephone_list
phone_number VARCHAR(8)
parent_id INT
Indexes

SQL Snippets

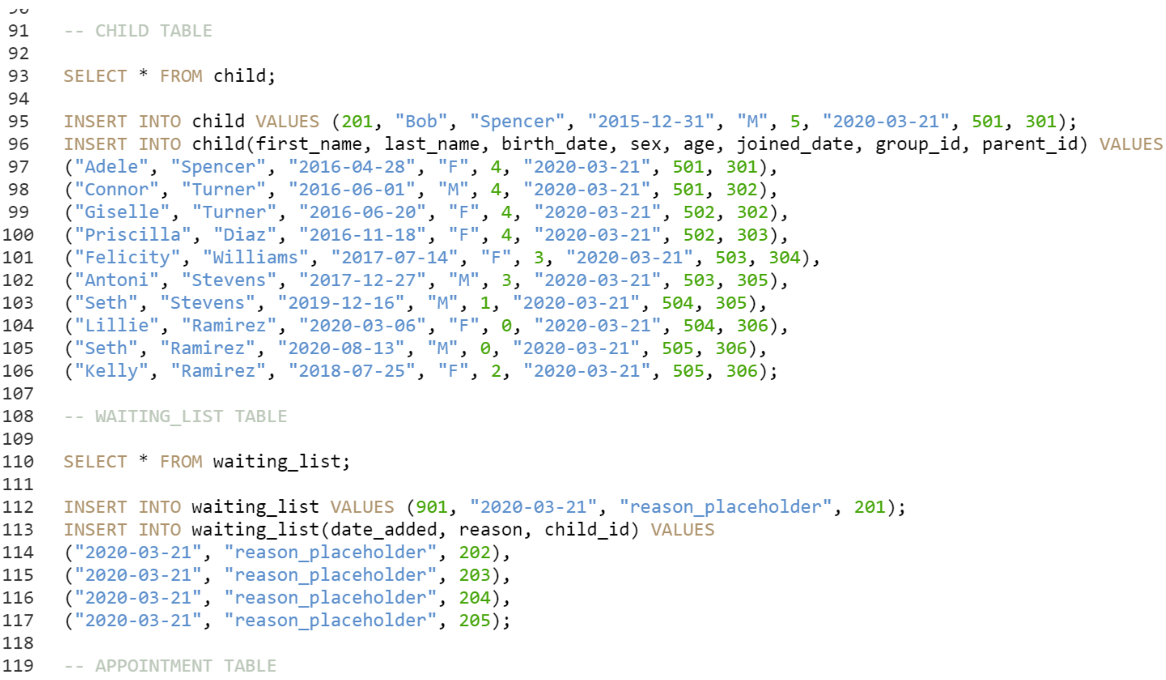
Roskilde Creation



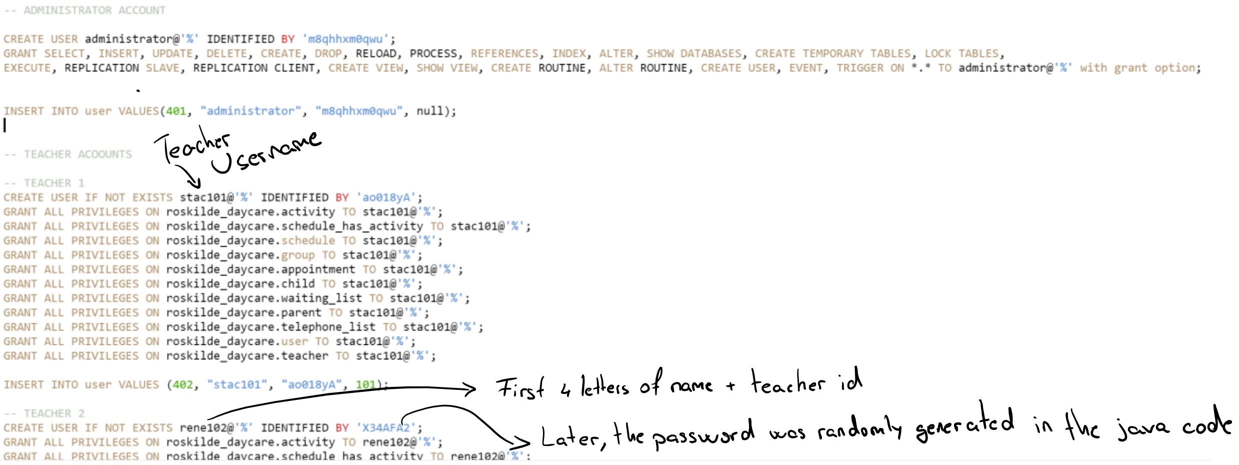
```sql
-- -----------------------------------------------------
-- Table `roskilde_daycare`.`group`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `roskilde_daycare`.`group` ;

CREATE TABLE IF NOT EXISTS `roskilde_daycare`.`group` (
  `group_id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NULL,
  `avg_age` DECIMAL(2,1) NOT NULL,
  `schedule_id` INT NOT NULL,
  PRIMARY KEY (`group_id`),
  CONSTRAINT `fk_group_schedule1`
    FOREIGN KEY (`schedule_id`)
    REFERENCES `roskilde_daycare`.`schedule` (`schedule_id`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION)
  ENGINE = InnoDB;
-- UNIQUE INDEX `name_UNIQUE` (`name` ASC) VISIBLE)

-- -----------------------------------------------------
-- Table `roskilde_daycare`.`child`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `roskilde_daycare`.`child` ;

CREATE TABLE IF NOT EXISTS `roskilde_daycare`.`child` (
  `child_id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(20) NOT NULL,
  `last_name` VARCHAR(20) NOT NULL,
  `birth_date` DATE NOT NULL,
  `sex` VARCHAR(1) NULL,
  `age` INT NOT NULL,
  `joined_date` DATE NULL,
  `group_id` INT,
```

Roskilde Population

•••



```sql
-- CHILD TABLE

SELECT * FROM child;

INSERT INTO child VALUES (201, "Bob", "Spencer", "2015-12-31", "M", 5, "2020-03-21", 501, 301);
INSERT INTO child(first_name, last_name, birth_date, sex, age, joined_date, group_id, parent_id) VALUES
("Adele", "Spencer", "2016-04-28", "F", 4, "2020-03-21", 501, 301),
("Connor", "Turner", "2016-06-01", "M", 4, "2020-03-21", 501, 302),
("Giselle", "Turner", "2016-06-20", "F", 4, "2020-03-21", 502, 302),
("Priscilla", "Diaz", "2016-11-18", "F", 4, "2020-03-21", 502, 303),
("Felicity", "Williams", "2017-07-14", "F", 3, "2020-03-21", 503, 304),
("Antoni", "Stevens", "2017-12-27", "M", 3, "2020-03-21", 503, 305),
("Seth", "Stevens", "2019-12-16", "M", 1, "2020-03-21", 504, 305),
("Lillie", "Ramirez", "2020-03-06", "F", 0, "2020-03-21", 504, 306),
("Seth", "Ramirez", "2020-08-13", "M", 0, "2020-03-21", 505, 306),
("Kelly", "Ramirez", "2018-07-25", "F", 2, "2020-03-21", 505, 306);

-- WAITING_LIST TABLE

SELECT * FROM waiting_list;

INSERT INTO waiting_list VALUES (901, "2020-03-21", "reason_placeholder", 201);
INSERT INTO waiting_list(date_added, reason, child_id) VALUES
("2020-03-21", "reason_placeholder", 202),
("2020-03-21", "reason_placeholder", 203),
("2020-03-21", "reason_placeholder", 204),
("2020-03-21", "reason_placeholder", 205);

-- APPOINTMENT TABLE
```



```sql
-- ADMINISTRATOR ACCOUNT

CREATE USER administrator@'%' IDENTIFIED BY 'm8qhhxm0qwu';
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES,
EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER ON *.* TO administrator@'%' with grant option;


INSERT INTO user VALUES(401, "administrator", "m8qhhxm0qwu", null);

-- TEACHER ACOOUNTS

-- TEACHER 1
CREATE USER IF NOT EXISTS stac101@'%' IDENTIFIED BY 'ao018yA';
GRANT ALL PRIVILEGES ON roskilde_daycare.activity TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.schedule_has_activity TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.schedule TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.group TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.appointment TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.child TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.waiting_list TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.parent TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.telephone_list TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.user TO stac101@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.teacher TO stac101@'%';

INSERT INTO user VALUES (402, "stac101", "ao018yA", 101);

-- TEACHER 2
CREATE USER IF NOT EXISTS rene102@'%' IDENTIFIED BY 'X34AFA2';
GRANT ALL PRIVILEGES ON roskilde_daycare.activity TO rene102@'%';
GRANT ALL PRIVILEGES ON roskilde_daycare.schedule_has_activity TO rene102@'%';
```

Teacher Username

First 4 letters of name + teacher id

Later, the password was randomly generated in the java code