Vietnam National University HCMC

International University

School of Computer Science and Engineering

PROJECT REPORT

TOPIC 1: ANDROID TASK MONITORING

<u>Group 14:</u> Principles of Database Management

Member	ID	Contribution	%
Võ Anh Việt (leader)	ITITIU19243	Real application - backend and frontend, MySQL, ERD, relational database.	40%
Nguyễn Đức Anh Tài	ITITIU19204	SQL database, data in SQL, 5 questions and queries, login-signup frontend, ERD.	20%
Trần Quang Tùng	ITITIU19237	Check and modify SQL, ERD.	15%
Vũ Bá Hưng	ITITU19126	Java connects with SQL, 5 questions and queries, ERD.	15%
Nguyễn Thiên Bảo	ITITIU17035	Queries, ERD.	10%

Advisor: Dr. Nguyen Thi Thuy Loan

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 - Introduction about the topic.

This project was launched because we realized that the world changes too fast for humans to catch up with everything. People are too busy with personal daily tasks, as well as the tasks from companies. Smartphones are very popular all around the world nowadays. Therefore, people can easily approach a mobile application. Android task monitoring born with the mission helps people accomplish their goals in any tasks, increase productivity, as well as simplifying their lives. Today, users can manage their life from home to work with only a touch thanks to our application. They can remind themselves in daily life with our Personal feature, and collaborate in a team via our Team feature.

1.2 - Reasons for our team to choose this topic.

Since the world economy is changing rapidly towards industrialization and modernization, people have to implement more and more tasks every day. However, they are struggling with how to arrange them effectively and properly to reach a better working performance within a certain time. Therefore, we launched our project to maximize human working capability by supporting them to schedule tasks better and optimize their time usage in a day.

1.3 - Methods to research.

First, our team will together design an ERD conceptual schema. Next step, we will convert from this diagram to the relational database schema. Knowledge to accomplish these steps is contained in the PDM course at IU. We divide this project into 2 parts: SQL - Java and an application in real life. At this time, Anh Tài, Bá Hưng, Quang Tùng, and Thiên Bảo will create a SQL database and a Java form (chapter 2, 3). SQL and Java form are already taught in the lab section by our advisor. Anh Việt will create a real application which will be described specifically in chapter 5. In the real application, we find the best suitable modal to create a complete app. Next step, we work with the backend which includes a server and a database. Then, the frontend will be designed and connected to all the things which the backend supplied.

1.4 - Project goals

1) Handling data and managing the database.
- Creating and installing the database.
- Designing models in the database.
- SQL language.
2) Understanding practical applications and real-life issues.
- Connecting between Java and databases.
- Satisfying all the requirements in a real application.
- Using databases to solve practical problems.
3) Creating a real application and making it work.

CHAPTER 2: ANALYSIS

2.1 - System Requirements.

Our project is designed for personal and teamwork purposes. The table below will describe all requirements for each purpose.

Requirements	Personal Tasks	Team Tasks
1	Register, log in and log out to account	
2		Create new teams (a team needs a manager) by default the user who created the team will be the manager.
3		Manage teams (users can see the list of teams they are in).
4		Add new users to a team.
5	Add new tasks	
	Users can add a new task with title, details, starting time - date, due time - date.	Users in a team can create any task and assign it to another user in the same team.
6	View the list of tasks (list of titles)	
	Our list of tasks will appear in order. Only the owners can see their tasks. They cannot see the tasks of other accounts.	In a team, the manager can see full the list of tasks in his/her team. Others can only see tasks to which they are allocated.
7	See the detailed information of tasks (details, starting date-time, due date-time)	
8	Update the progress of tasks (done or in progress)	
9	Delete tasks	
	Users can delete any task they want.	Only the manager can delete tasks in the team.

2.2 - Entity Relationship Diagram (ERD)

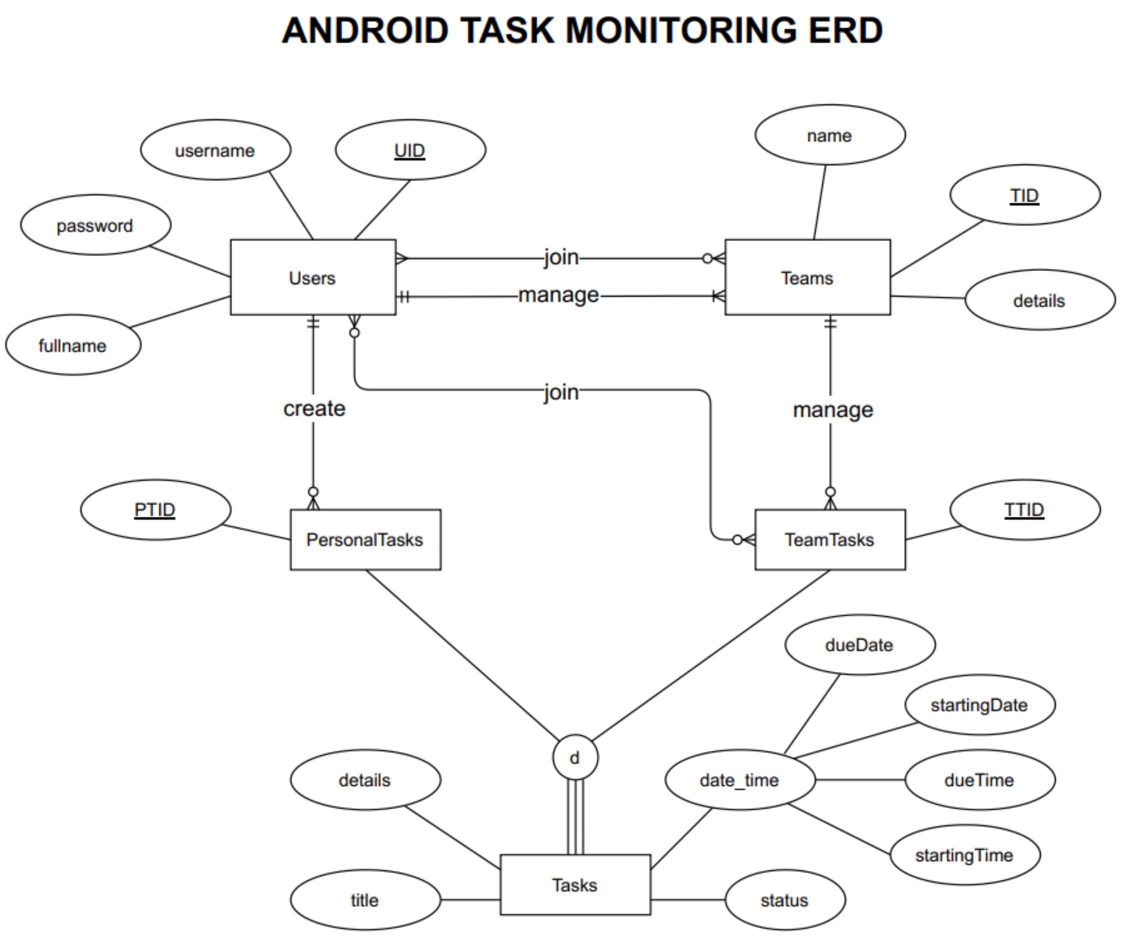


Figure 2.1: Entity Relationship Diagram by draw.io

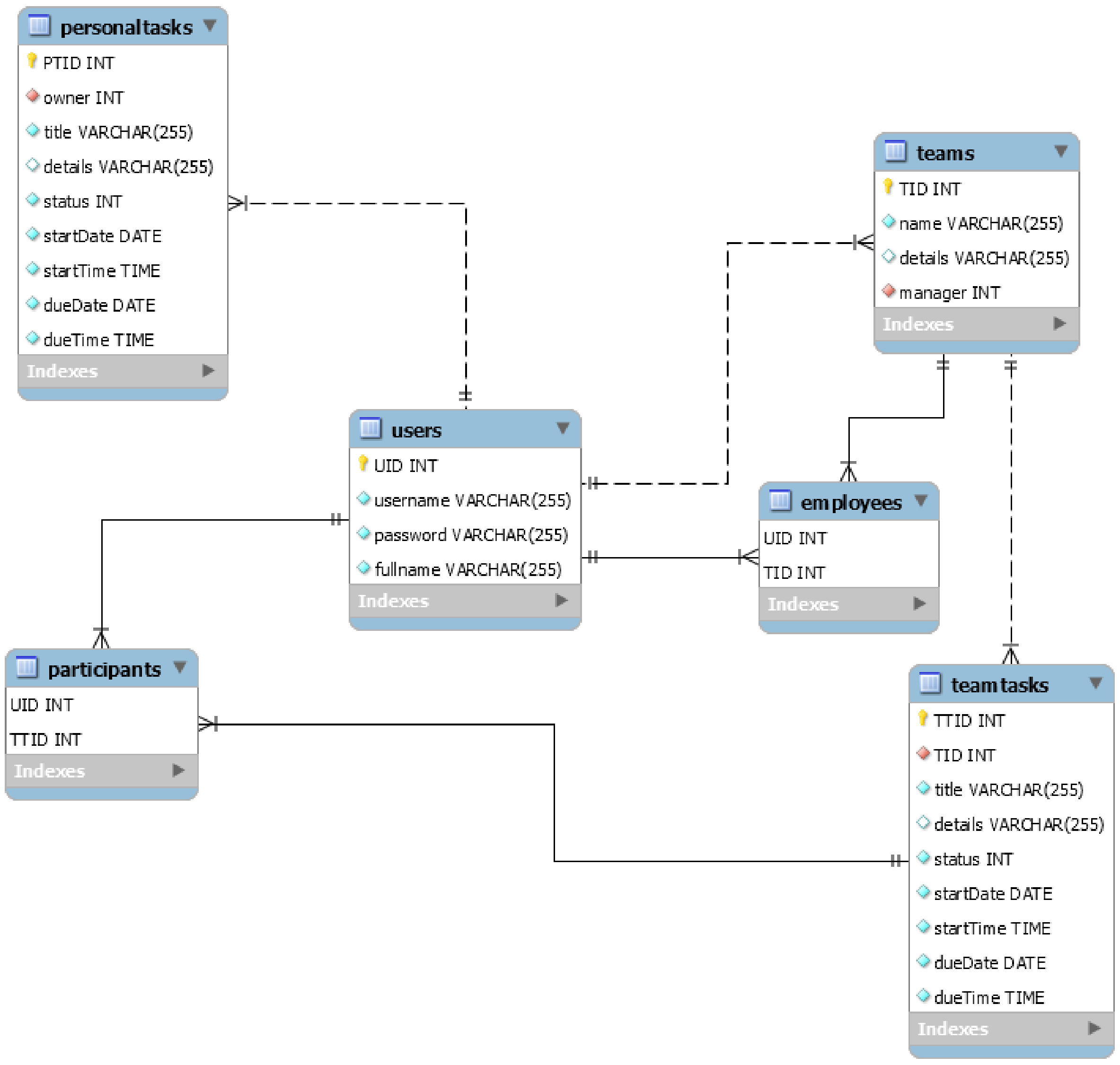


Figure 2.2: Entity Relationship Diagram by MySQL

2.3 - Relational Database Schema

Users-schema = (<u>UID</u>, username, password, fullname)

PersonalTasks-schema = (<u>PTID</u>, owner, title, details, status, startDate, StartTime, dueDate, dueTime)

- From PersonalTasks-schema.owner to Users-schema.UID

Teams-schema = (<u>TID</u>, name, details, manager)

- From Teams-schema.manager to Users-schema.UID

Employees-schema = (<u>UID, TID</u>)

- From Employees-schema.UID to Users-schema.UID
- From Employees-schema.TID to Teams-schema.TID

TeamTasks-schema = (TTID, TID, title, details, status, startDate, startTime, dueDate, dueTime)

- From TeamTasks-schema.TID to Teams-schema.TID

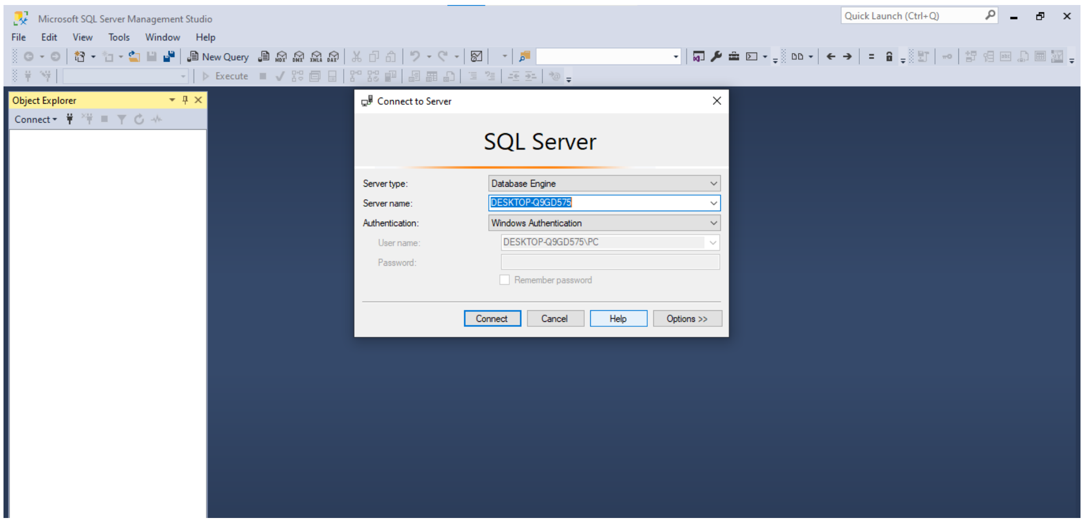
Participants-schema = (TTID, UID);

- From Participants-schema.UID to Users-schema.UID
- From Participants-schema.TTID to TeamTasks-schema.TTID.

CHAPTER 3: IMPLEMENT ON SQL

3.1 - Create Database

- With the database, my group uses the SQL Server Management Studio graphical interface to create a database and this is how we create a database :
  + First, we have registered and connected to my server.



Server type: Database Engine
Server name: DESKTOP-Q9GD575
Authentication: Windows Authentication
User name: DESKTOP-Q9GD575\PC
Password:
Remember password

Figure 3.1: Connect to SQL Server

  + After connecting to my server, in object explorer, we will see the database node, expand it to see system databases and database snapshot node to ensure everything is ready for the next action.

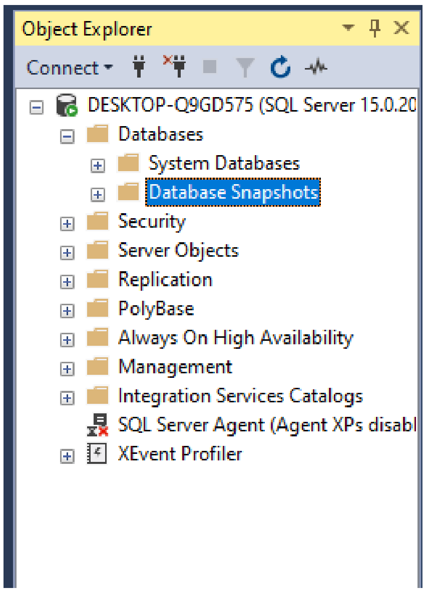


Figure 3.2: Manage databases by SQL Server

  + Next, right-click the Databases node to bring up a pop-up menu with several different options. Choose “New Database”.

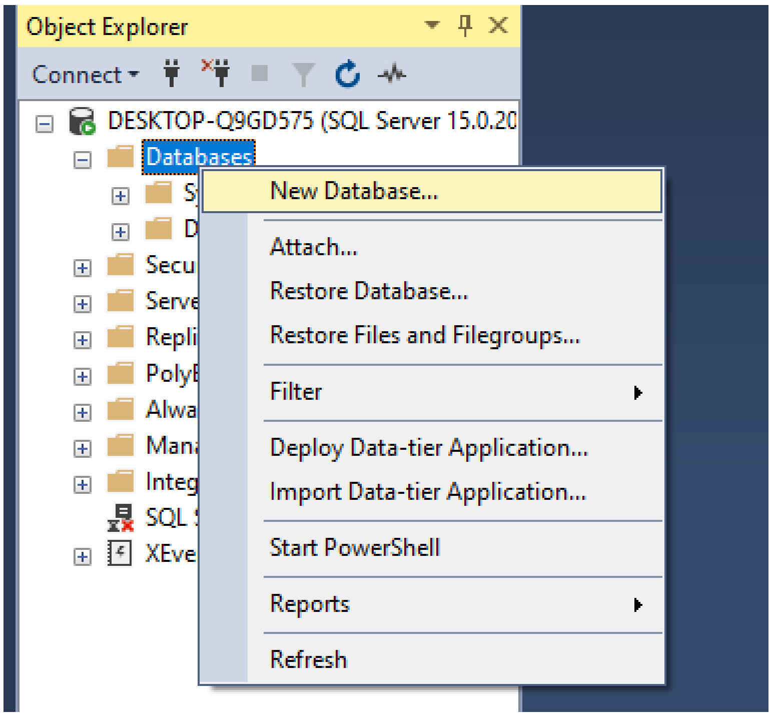
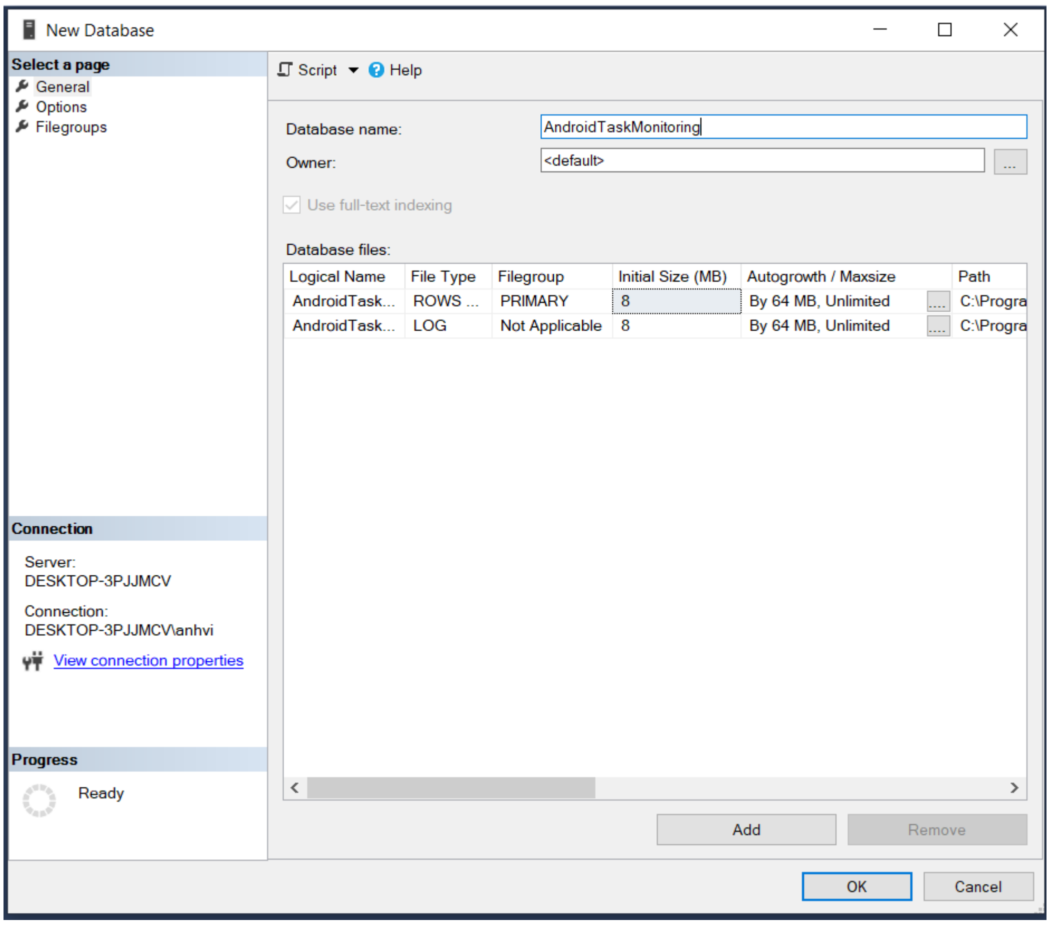


Figure 3.3: Create new database

+ We created my database's name is AndroidTaskMonitoring.



Database name: AndroidTaskMonitoring

Owner: <default>

Use full-text indexing

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize	Path
AndroidTask...	ROWS ...	PRIMARY	8	By 64 MB, Unlimited	C:\Progra
AndroidTask...	LOG	Not Applicable	8	By 64 MB, Unlimited	C:\Progra

Connection

Server:
DESKTOP-3PJJMCV

Connection:
DESKTOP-3PJJMCV\anhvi

View connection properties

Progress

Ready

Figure 3.4: A new database with name AndroidTaskMonitoring

+ Back in the Object Explorer in SQL Server Management Studio, the new database is listed on the Database node.

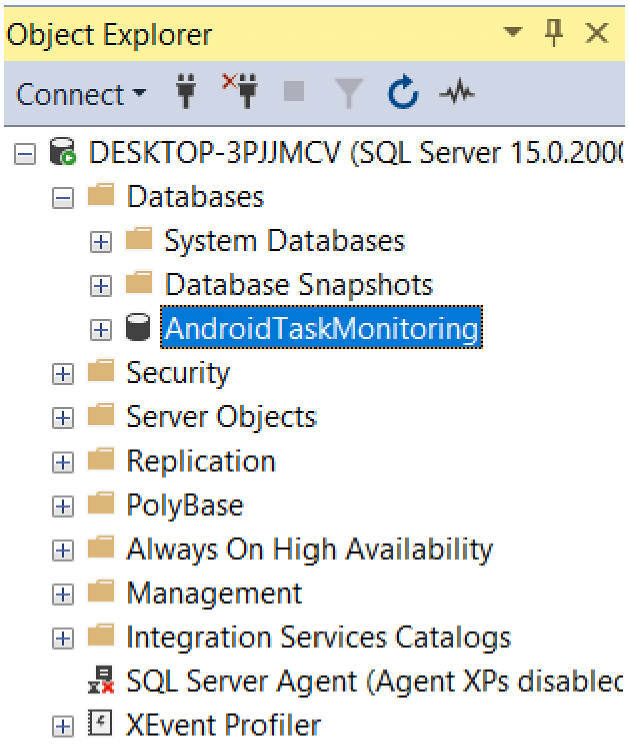
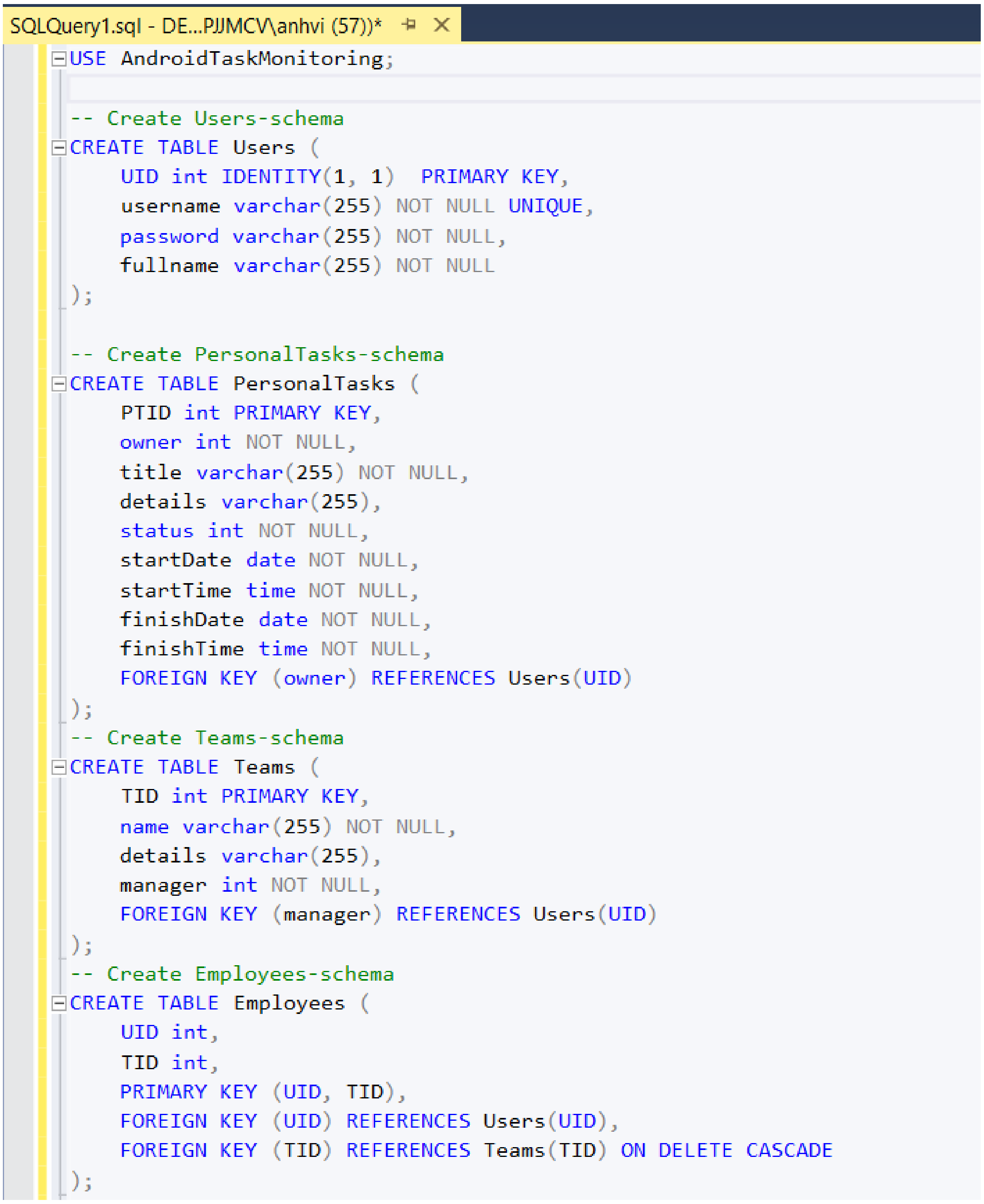


Figure 3.5: Create a new database successfully

3.2 - Input Data And Testing By SQL

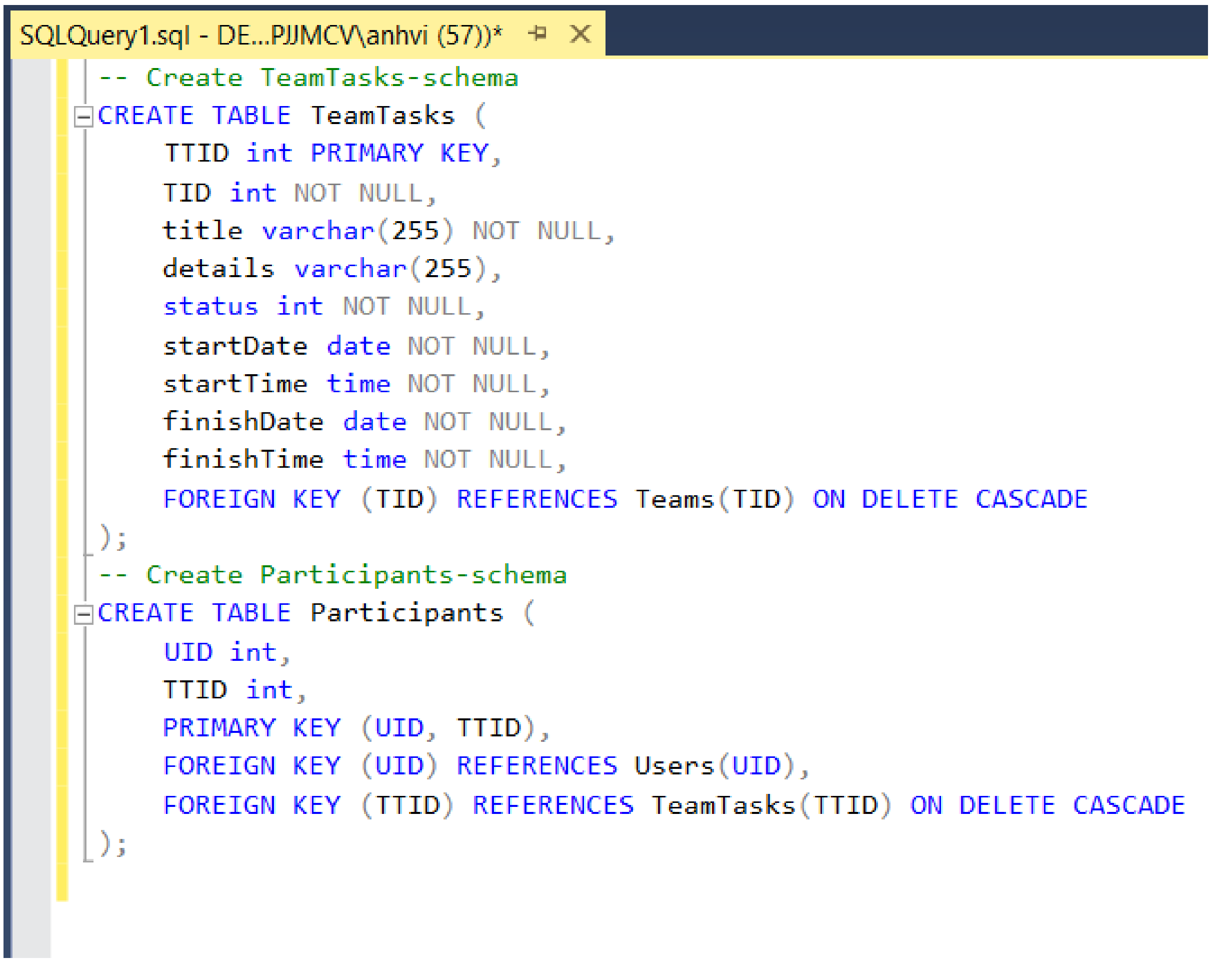
- After completely creating the Database, we will input the data on our database by Query.

+ First, we need to create 6 tables that are Users, Teams, PersonalTasks, TeamTasks, Employees, and Participants.



```sql
USE AndroidTaskMonitoring;

-- Create Users-schema
CREATE TABLE Users (
    UID int IDENTITY(1, 1)  PRIMARY KEY,
    username varchar(255) NOT NULL UNIQUE,
    password varchar(255) NOT NULL,
    fullname varchar(255) NOT NULL
);

-- Create PersonalTasks-schema
CREATE TABLE PersonalTasks (
    PTID int PRIMARY KEY,
    owner int NOT NULL,
    title varchar(255) NOT NULL,
    details varchar(255),
    status int NOT NULL,
    startDate date NOT NULL,
    startTime time NOT NULL,
    finishDate date NOT NULL,
    finishTime time NOT NULL,
    FOREIGN KEY (owner) REFERENCES Users(UID)
);
-- Create Teams-schema
CREATE TABLE Teams (
    TID int PRIMARY KEY,
    name varchar(255) NOT NULL,
    details varchar(255),
    manager int NOT NULL,
    FOREIGN KEY (manager) REFERENCES Users(UID)
);
-- Create Employees-schema
CREATE TABLE Employees (
    UID int,
    TID int,
    PRIMARY KEY (UID, TID),
    FOREIGN KEY (UID) REFERENCES Users(UID),
    FOREIGN KEY (TID) REFERENCES Teams(TID) ON DELETE CASCADE
);
```

Figure 3.6: Users, PersonalTasks, Teams, and Employees table



SQLQuery1.sql - DE...PJJMCV\anhvi (57))*

```
-- Create TeamTasks-schema
CREATE TABLE TeamTasks (
    TTID int PRIMARY KEY,
    TID int NOT NULL,
    title varchar(255) NOT NULL,
    details varchar(255),
    status int NOT NULL,
    startDate date NOT NULL,
    startTime time NOT NULL,
    finishDate date NOT NULL,
    finishTime time NOT NULL,
    FOREIGN KEY (TID) REFERENCES Teams(TID) ON DELETE CASCADE
);
-- Create Participants-schema
CREATE TABLE Participants (
    UID int,
    TTID int,
    PRIMARY KEY (UID, TTID),
    FOREIGN KEY (UID) REFERENCES Users(UID),
    FOREIGN KEY (TTID) REFERENCES TeamTasks(TTID) ON DELETE CASCADE
);
```

Figure 3.7: TeamTasks and Participants table

+ This is a diagram for our database.

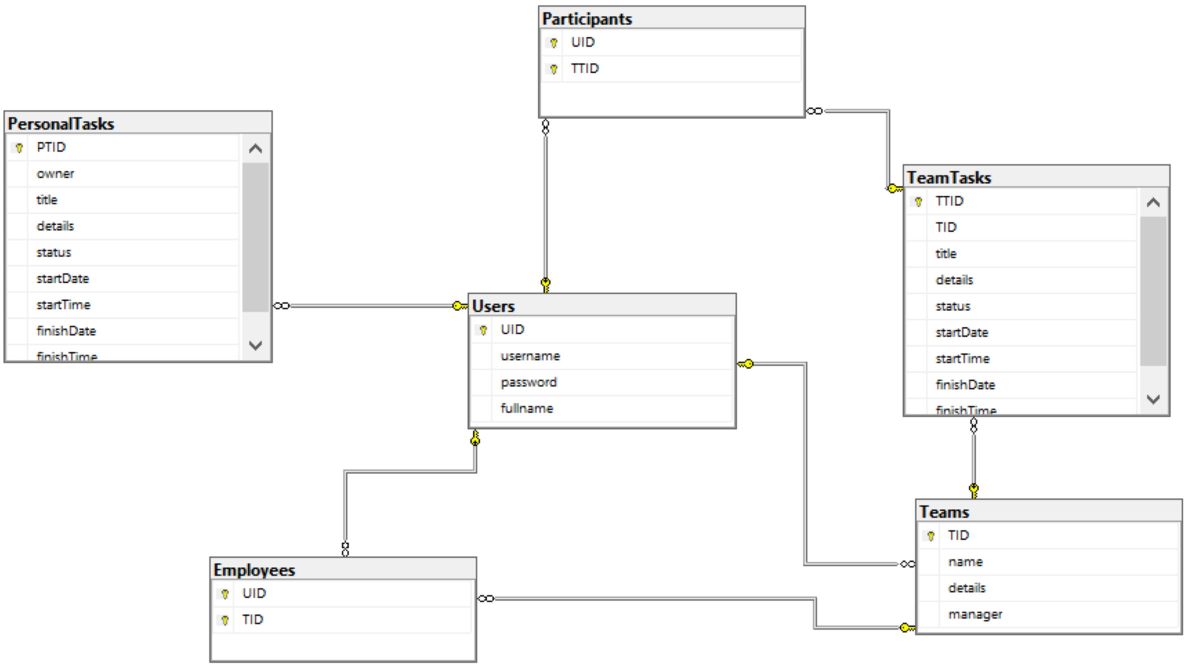
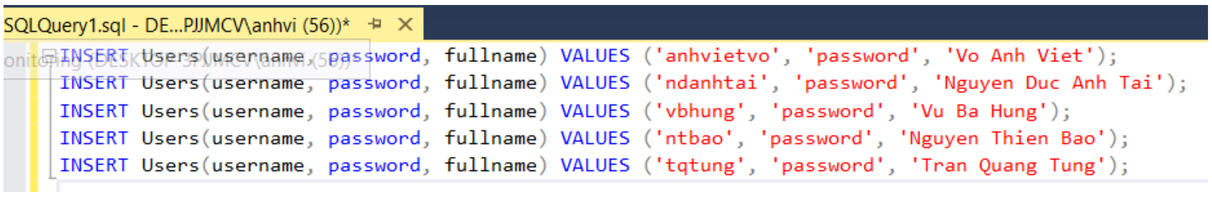


Figure 3.8: Database diagram by SQL Server

Then, we need to insert data into table Users, Teams, PersonalTasks, TeamTasks, Employees and Participants.

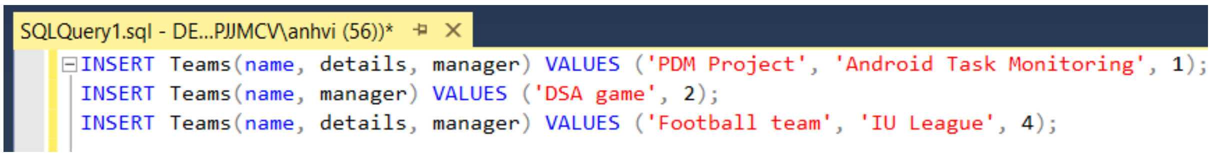
Users:



```sql
INSERT Users(username, password, fullname) VALUES ('anhvietvo', 'password', 'Vo Anh Viet');
INSERT Users(username, password, fullname) VALUES ('ndanhtai', 'password', 'Nguyen Duc Anh Tai');
INSERT Users(username, password, fullname) VALUES ('vbhung', 'password', 'Vu Ba Hung');
INSERT Users(username, password, fullname) VALUES ('ntbao', 'password', 'Nguyen Thien Bao');
INSERT Users(username, password, fullname) VALUES ('tqtung', 'password', 'Tran Quang Tung');
```

Figure 3.9: Insert values into Users table

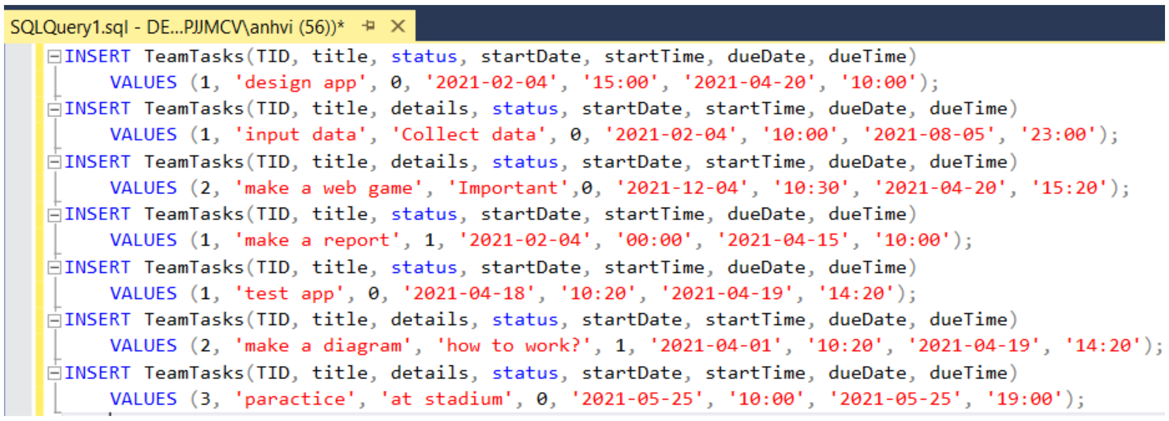
Teams:



```sql
INSERT Teams(name, details, manager) VALUES ('PDM Project', 'Android Task Monitoring', 1);
INSERT Teams(name, manager) VALUES ('DSA game', 2);
INSERT Teams(name, details, manager) VALUES ('Football team', 'IU League', 4);
```

Figure 3.10: Insert values into Teams table

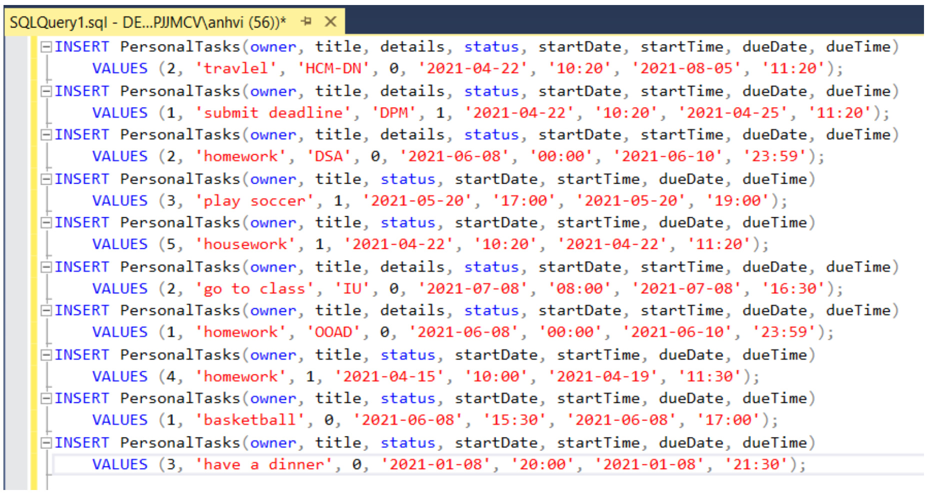
TeamTasks:



```sql
INSERT TeamTasks(TID, title, status, startDate, startTime, dueDate, dueTime)
    VALUES (1, 'design app', 0, '2021-02-04', '15:00', '2021-04-20', '10:00');
INSERT TeamTasks(TID, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (1, 'input data', 'Collect data', 0, '2021-02-04', '10:00', '2021-08-05', '23:00');
INSERT TeamTasks(TID, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (2, 'make a web game', 'Important',0, '2021-12-04', '10:30', '2021-04-20', '15:20');
INSERT TeamTasks(TID, title, status, startDate, startTime, dueDate, dueTime)
    VALUES (1, 'make a report', 1, '2021-02-04', '00:00', '2021-04-15', '10:00');
INSERT TeamTasks(TID, title, status, startDate, startTime, dueDate, dueTime)
    VALUES (1, 'test app', 0, '2021-04-18', '10:20', '2021-04-19', '14:20');
INSERT TeamTasks(TID, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (2, 'make a diagram', 'how to work?', 1, '2021-04-01', '10:20', '2021-04-19', '14:20');
INSERT TeamTasks(TID, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (3, 'paractice', 'at stadium', 0, '2021-05-25', '10:00', '2021-05-25', '19:00');
```

Figure 3.11: Insert values into TeamTasks table

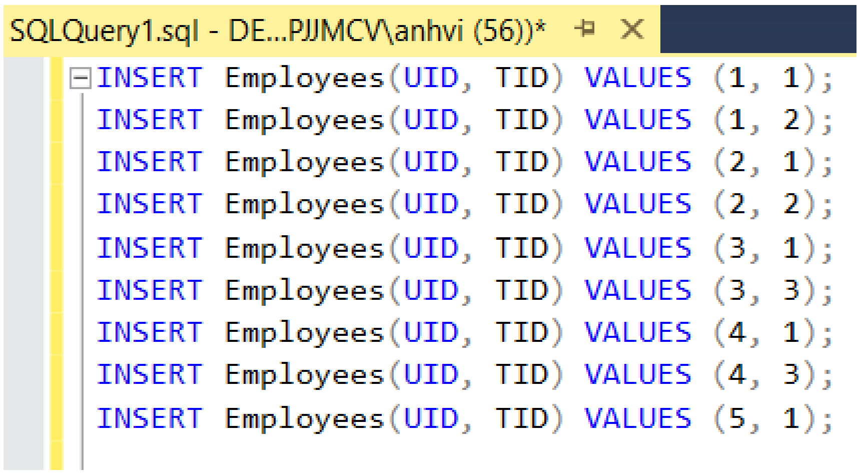
PersonalTasks:



```sql
INSERT PersonalTasks(owner, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (2, 'travlel', 'HCM-DN', 0, '2021-04-22', '10:20', '2021-08-05', '11:20');
INSERT PersonalTasks(owner, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (1, 'submit deadline', 'DPM', 1, '2021-04-22', '10:20', '2021-04-25', '11:20');
INSERT PersonalTasks(owner, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (2, 'homework', 'DSA', 0, '2021-06-08', '00:00', '2021-06-10', '23:59');
INSERT PersonalTasks(owner, title, status, startDate, startTime, dueDate, dueTime)
    VALUES (3, 'play soccer', 1, '2021-05-20', '17:00', '2021-05-20', '19:00');
INSERT PersonalTasks(owner, title, status, startDate, startTime, dueDate, dueTime)
    VALUES (5, 'housework', 1, '2021-04-22', '10:20', '2021-04-22', '11:20');
INSERT PersonalTasks(owner, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (2, 'go to class', 'IU', 0, '2021-07-08', '08:00', '2021-07-08', '16:30');
INSERT PersonalTasks(owner, title, details, status, startDate, startTime, dueDate, dueTime)
    VALUES (1, 'homework', 'OOAD', 0, '2021-06-08', '00:00', '2021-06-10', '23:59');
INSERT PersonalTasks(owner, title, status, startDate, startTime, dueDate, dueTime)
    VALUES (4, 'homework', 1, '2021-04-15', '10:00', '2021-04-19', '11:30');
INSERT PersonalTasks(owner, title, status, startDate, startTime, dueDate, dueTime)
    VALUES (1, 'basketball', 0, '2021-06-08', '15:30', '2021-06-08', '17:00');
INSERT PersonalTasks(owner, title, status, startDate, startTime, dueDate, dueTime)
    VALUES (3, 'have a dinner', 0, '2021-01-08', '20:00', '2021-01-08', '21:30');
```

Figure 3.12: Insert values into PersonalTasks table

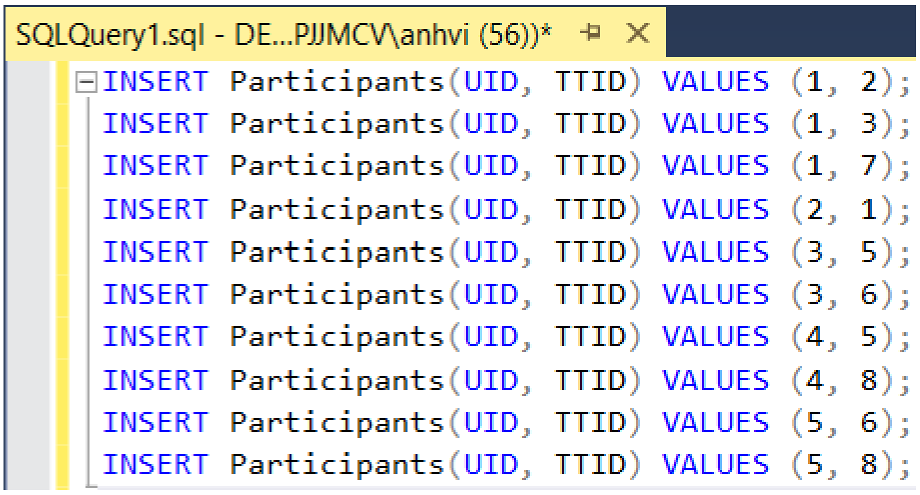
Employees:


SQLQuery1.sql - DE...PJJMCV\anhvi (56))*

```
INSERT Employees(UID, TID) VALUES (1, 1);
INSERT Employees(UID, TID) VALUES (1, 2);
INSERT Employees(UID, TID) VALUES (2, 1);
INSERT Employees(UID, TID) VALUES (2, 2);
INSERT Employees(UID, TID) VALUES (3, 1);
INSERT Employees(UID, TID) VALUES (3, 3);
INSERT Employees(UID, TID) VALUES (4, 1);
INSERT Employees(UID, TID) VALUES (4, 3);
INSERT Employees(UID, TID) VALUES (5, 1);
```

Figure 3.13: Insert values into Employees table

Participants:

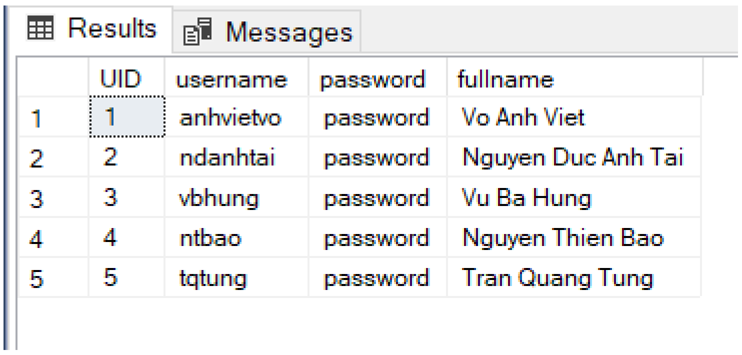

SQLQuery1.sql - DE...PJJMCV\anhvi (56))*

```
INSERT Participants(UID, TTID) VALUES (1, 2);
INSERT Participants(UID, TTID) VALUES (1, 3);
INSERT Participants(UID, TTID) VALUES (1, 7);
INSERT Participants(UID, TTID) VALUES (2, 1);
INSERT Participants(UID, TTID) VALUES (3, 5);
INSERT Participants(UID, TTID) VALUES (3, 6);
INSERT Participants(UID, TTID) VALUES (4, 5);
INSERT Participants(UID, TTID) VALUES (4, 8);
INSERT Participants(UID, TTID) VALUES (5, 6);
INSERT Participants(UID, TTID) VALUES (5, 8);
```

Figure 3.14: Insert values into Participants table

+ These are datas in each table.

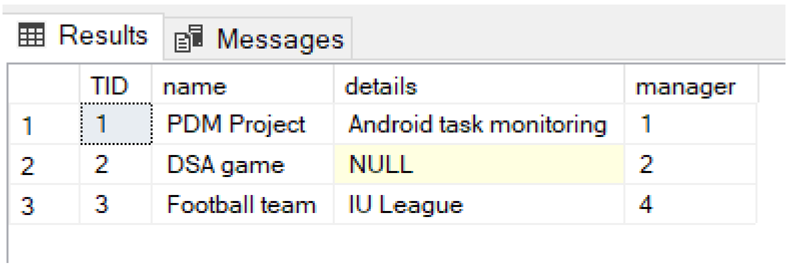
Users:



	UID	username	password	fullname
1	1	anhvietvo	password	Vo Anh Viet
2	2	ndanhtai	password	Nguyen Duc Anh Tai
3	3	vbhung	password	Vu Ba Hung
4	4	ntbao	password	Nguyen Thien Bao
5	5	tqtung	password	Tran Quang Tung

Figure 3.15: Data in Users table

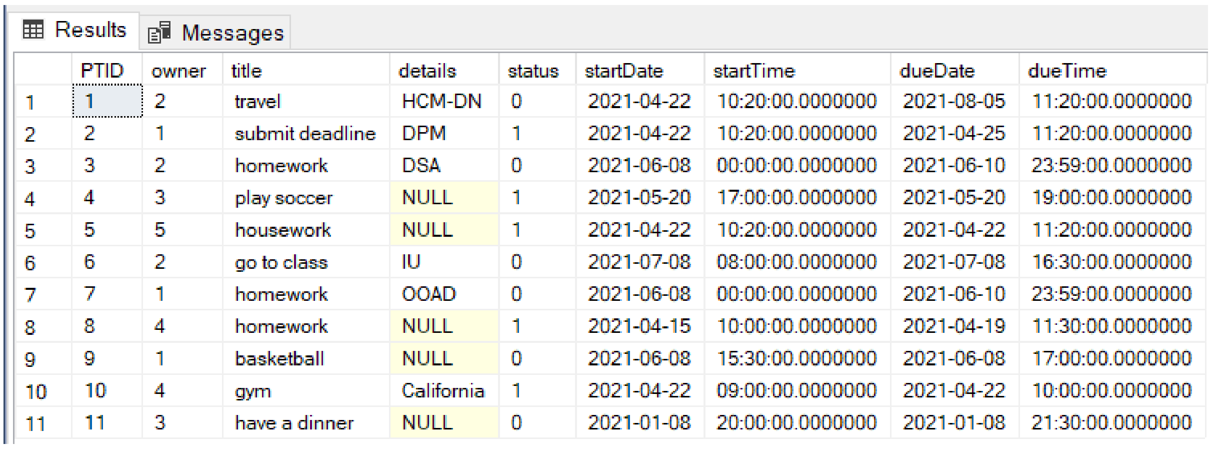
Teams:



	TID	name	details	manager
1	1	PDM Project	Android task monitoring	1
2	2	DSA game	NULL	2
3	3	Football team	IU League	4

Figure 3.16: Data in Teams table

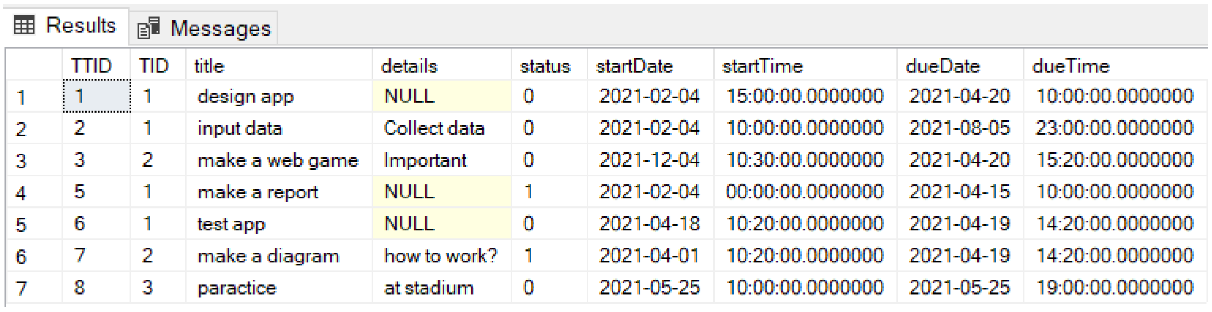
PersonalTasks:



	PTID	owner	title	details	status	startDate	startTime	dueDate	dueTime
1	1	2	travel	HCM-DN	0	2021-04-22	10:20:00.0000000	2021-08-05	11:20:00.0000000
2	2	1	submit deadline	DPM	1	2021-04-22	10:20:00.0000000	2021-04-25	11:20:00.0000000
3	3	2	homework	DSA	0	2021-06-08	00:00:00.0000000	2021-06-10	23:59:00.0000000
4	4	3	play soccer	NULL	1	2021-05-20	17:00:00.0000000	2021-05-20	19:00:00.0000000
5	5	5	housework	NULL	1	2021-04-22	10:20:00.0000000	2021-04-22	11:20:00.0000000
6	6	2	go to class	IU	0	2021-07-08	08:00:00.0000000	2021-07-08	16:30:00.0000000
7	7	1	homework	OOAD	0	2021-06-08	00:00:00.0000000	2021-06-10	23:59:00.0000000
8	8	4	homework	NULL	1	2021-04-15	10:00:00.0000000	2021-04-19	11:30:00.0000000
9	9	1	basketball	NULL	0	2021-06-08	15:30:00.0000000	2021-06-08	17:00:00.0000000
10	10	4	gym	California	1	2021-04-22	09:00:00.0000000	2021-04-22	10:00:00.0000000
11	11	3	have a dinner	NULL	0	2021-01-08	20:00:00.0000000	2021-01-08	21:30:00.0000000

Figure 3.17: Data in PersonalTasks table

TeamTasks:



	TTID	TID	title	details	status	startDate	startTime	dueDate	dueTime
1	1	1	design app	NULL	0	2021-02-04	15:00:00.0000000	2021-04-20	10:00:00.0000000
2	2	1	input data	Collect data	0	2021-02-04	10:00:00.0000000	2021-08-05	23:00:00.0000000
3	3	2	make a web game	Important	0	2021-12-04	10:30:00.0000000	2021-04-20	15:20:00.0000000
4	5	1	make a report	NULL	1	2021-02-04	00:00:00.0000000	2021-04-15	10:00:00.0000000
5	6	1	test app	NULL	0	2021-04-18	10:20:00.0000000	2021-04-19	14:20:00.0000000
6	7	2	make a diagram	how to work?	1	2021-04-01	10:20:00.0000000	2021-04-19	14:20:00.0000000
7	8	3	paractice	at stadium	0	2021-05-25	10:00:00.0000000	2021-05-25	19:00:00.0000000

Figure 3.18: Data in TeamTasks table

3.3 - List Of Questions And Answer By SQL

1. Find the teamName, taskID,accountName who haven't completed all the tasks in teamName ='DSA game' with dueDate = '20/04/2021'.

```
SELECT  name, TTID, title
FROM Teams INNER JOIN TeamTasks
ON  TeamTasks.TID = Teams.TID
WHERE TeamTasks.status = 0 AND dueDate = '04-20-21' AND teamName = 'DSA game'
```

2. Find the username and fullname of members who join in both teamName = ‘PDM project’ and teamName = ‘DSA game’.

```
SELECT fullname, username
FROM  Users INNER  JOIN Employees
        ON  Users.UID = Employees.UID
WHERE Employees.TID IN  (SELECT TID FROM Teams WHERE name = 'PDM Project')
        INTERSECT
SELECT fullname, username
FROM Users INNER JOIN Employees
        ON Users.UID = Employees.UID
WHERE Employees.TID IN (SELECT TID FROM Teams WHERE name = 'DSA game')
```

3. Lists the number of tasks in each team. Only include teams with more than 3 tasks.

```
SELECT  Teams.name, COUNT(TeamTasks.TTID) AS tasksnumber
FROM TeamTasks
        INNER  JOIN teams
        ON  TeamTasks.TID = Teams.TID
        GROUP BY name
        HAVING COUNT(TeamTasks.TTID) > 3
```

4. Lists top 3 fullname, the number of tasks that haven't been completed by users in Personal Tasks in descending order.

```
SELECT TOP 3  Users.fullname , COUNT(PersonalTasks.Status ) AS total_not_complete
FROM   PersonalTasks INNER JOIN Users
        ON Users.UID = PersonalTasks.owner
WHERE status = '0'
GROUP BY Users.fullname
ORDER BY total_not_complete DESC
```

5. List fullname, number of taskId of users who completed tasks in personal tasks from 10/05/2021 to 08/06/2021.

```
SELECT Users.fullname, COUNT(personalTasks.PTID) AS total_task
FROM PersonalTasks
        INNER JOIN Users
        ON Users.UID = PersonalTasks.owner
WHERE personalTasks.dueDate >= '2021-05-10'
        AND   personalTasks.dueDate <= '2021-06-08' AND personalTasks.status = '1'
GROUP BY users.fullname
```

6. Find Team name and number of tasks for each team in descending order

```
SELECT      t2.TID, COUNT(TTID) AS Number_of_tasks
FROM teams t1, teamTasks t2
WHERE t1.TID = t2.TID
GROUP BY t2.TID
ORDER BY COUNT(TTID) DESC
```

7. List username, taskId, title of personalTasks according to the amount of time of user’s tasks in descending order.

```
SELECT      username, PTID , title, DATEDIFF(HOUR, startingDate, dueDate) +
DATEDIFF(HOUR, startTime, dueTime) As hour_duration
FROM users, personalTasks
WHERE users.UID = personalTasks.owner
ORDER BY hour_duration desc
```

8. Find the number of the tasks = ‘homework’ or ‘play soccer’ in personalTasks

```
SELECT title, count(title) as Quantity
FROM  personalTasks T
WHERE  T.title = 'play soccer' or T.title = 'homework'
group by title
```

9. List all tasks were completed in TeamTasks and the date/time for the task was completed.

```
SELECT name, title, startDate, dueDate
FROM TeamTasks t1, Teams t2
WHERE t1.TID = t2.TID
        AND dueDate < (SELECT CAST(GETDATE() AS date)) AND t1.status = 1
ORDER BY name
```

CHAPTER 4: CONNECTION FROM JAVA TO DATABASE

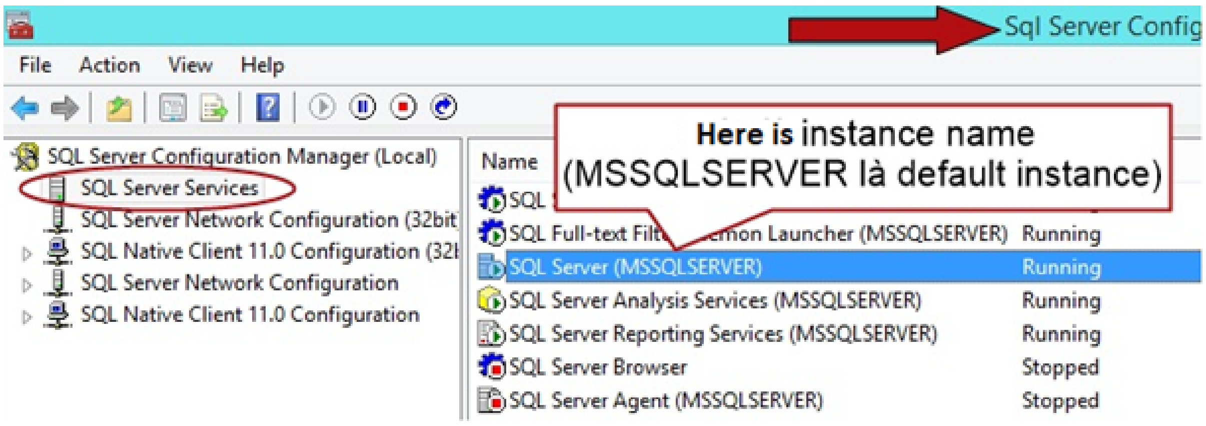
4.1 - Database Setting

1. Create a login user

- SQL Server JDBC connection string have this form:
  . jdbc:sqlserver://[serverName[\instanceName][:portNumber]][; property = value [; property = value]]

In which:

- ServerName is the name server or IP address of the server where Microsoft SQL Server is installed
- The instanceName of an instance to connect to serverName. If the this parameter is not defined only, then the default case will be used
  + Login to SQL Server with Authentication is set to **Windows Authentication**.
  + Find localport of sql server by **SQL SERVER CONFIGURATION**:

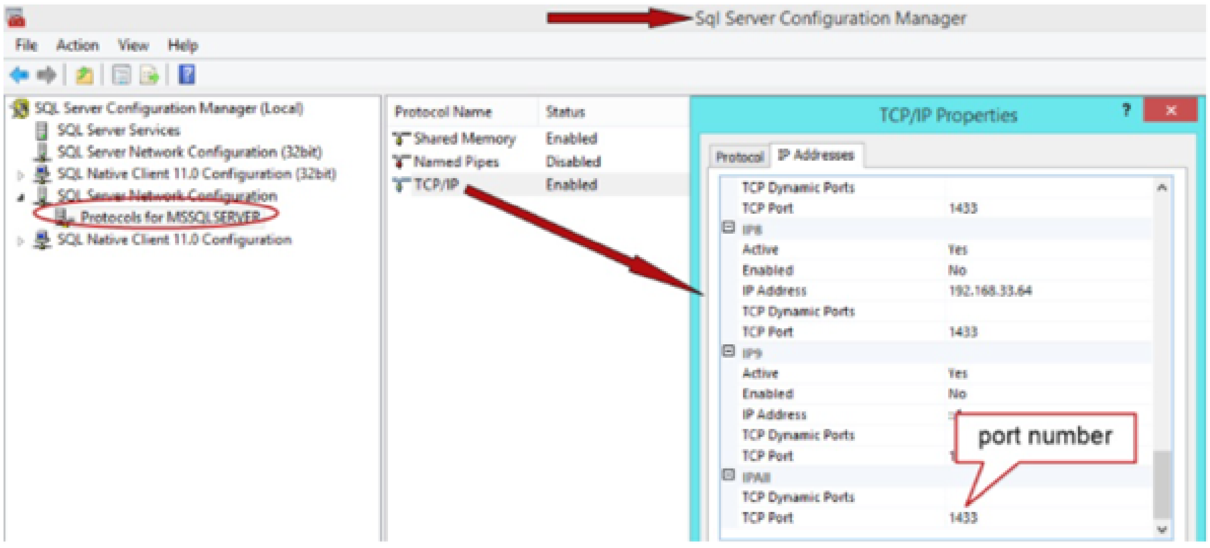


SQL Server Configuration Manager (Local)
- SQL Server Services
- SQL Server Network Configuration (32bit
- SQL Native Client 11.0 Configuration (32t
- SQL Server Network Configuration
- SQL Native Client 11.0 Configuration

Name	
SQL	
SQL Full-text Filt...mon Launcher (MSSQLSERVER)	Running
SQL Server (MSSQLSERVER)	Running
SQL Server Analysis Services (MSSQLSERVER)	Running
SQL Server Reporting Services (MSSQLSERVER)	Running
SQL Server Browser	Stopped
SQL Server Agent (MSSQLSERVER)	Stopped

Figure 4.1: SQL Server configuration

- Port number is the SQL Server application port, by default is 1433.



SQL Server Configuration Manager (Local)
- SQL Server Services
- SQL Server Network Configuration (32bit)
- SQL Native Client 11.0 Configuration (32bit)
- SQL Server Network Configuration
  - Protocols for MSSQLSERVER
- SQL Native Client 11.0 Configuration

Protocol Name	Status
Shared Memory	Enabled
Named Pipes	Disabled
TCP/IP	Enabled

TCP Dynamic Ports	
TCP Port	1433
IP8	
Active	Yes
Enabled	No
IP Address	192.168.33.64
TCP Dynamic Ports	
TCP Port	1433
IP9	
Active	Yes
Enabled	No
IP Address	
TCP Dynamic Ports	
TCP Port	
IPAll	
TCP Dynamic Ports	
TCP Port	1433

Figure 4.2: Port number of SQL Server

- Property = value specifies the database name, SQL Server login name and password databaseName = Androidmonitoring; user = sa; password = sa.
- Setting account for **sa**:

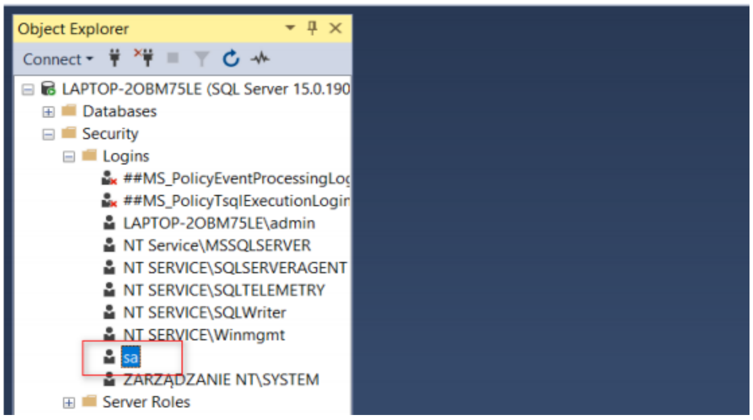


Figure 4.3: sa account in SQL

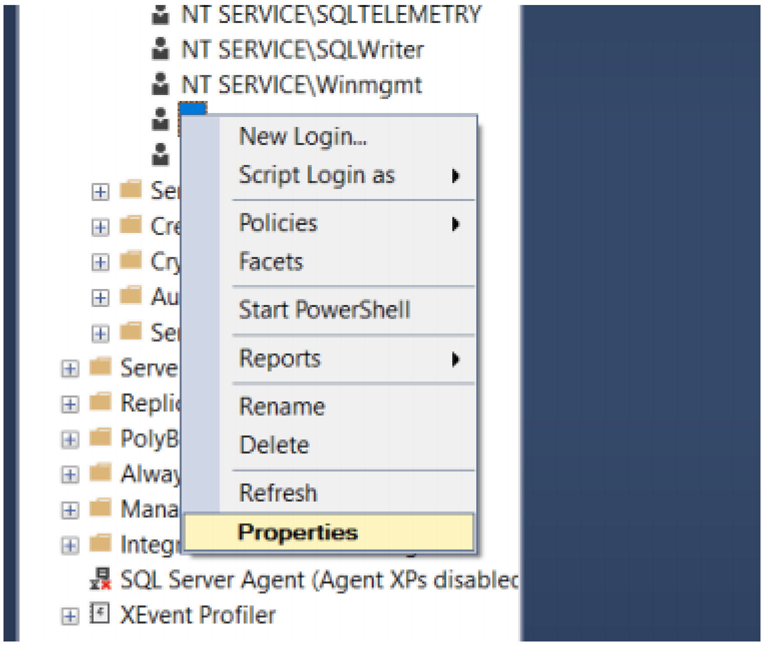
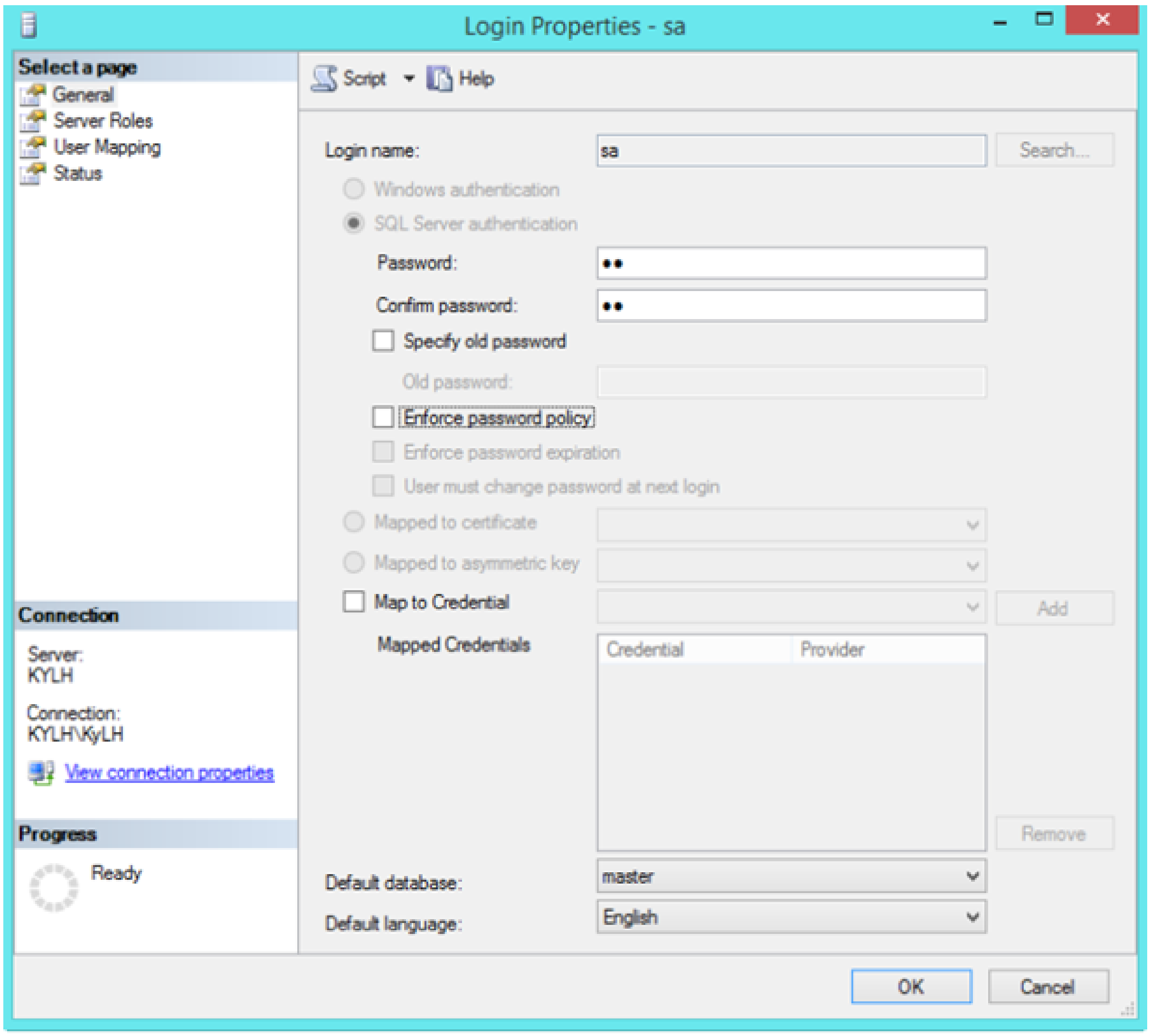


Figure 4.4: Find properties of sa account

- Next a new console appears and let setting password for **sa**.


Login Properties - sa

Select a page
- General
- Server Roles
- User Mapping
- Status

Connection

Server:
KYLH

Connection:
KYLH\KyLH

View connection properties

Progress

Ready

Script ▾ Help

Login name: sa

( ) Windows authentication
(●) SQL Server authentication

Password: ••
Confirm password: ••
[ ] Specify old password
Old password:
[ ] Enforce password policy
[ ] Enforce password expiration
[ ] User must change password at next login

( ) Mapped to certificate
( ) Mapped to asymmetric key
[ ] Map to Credential

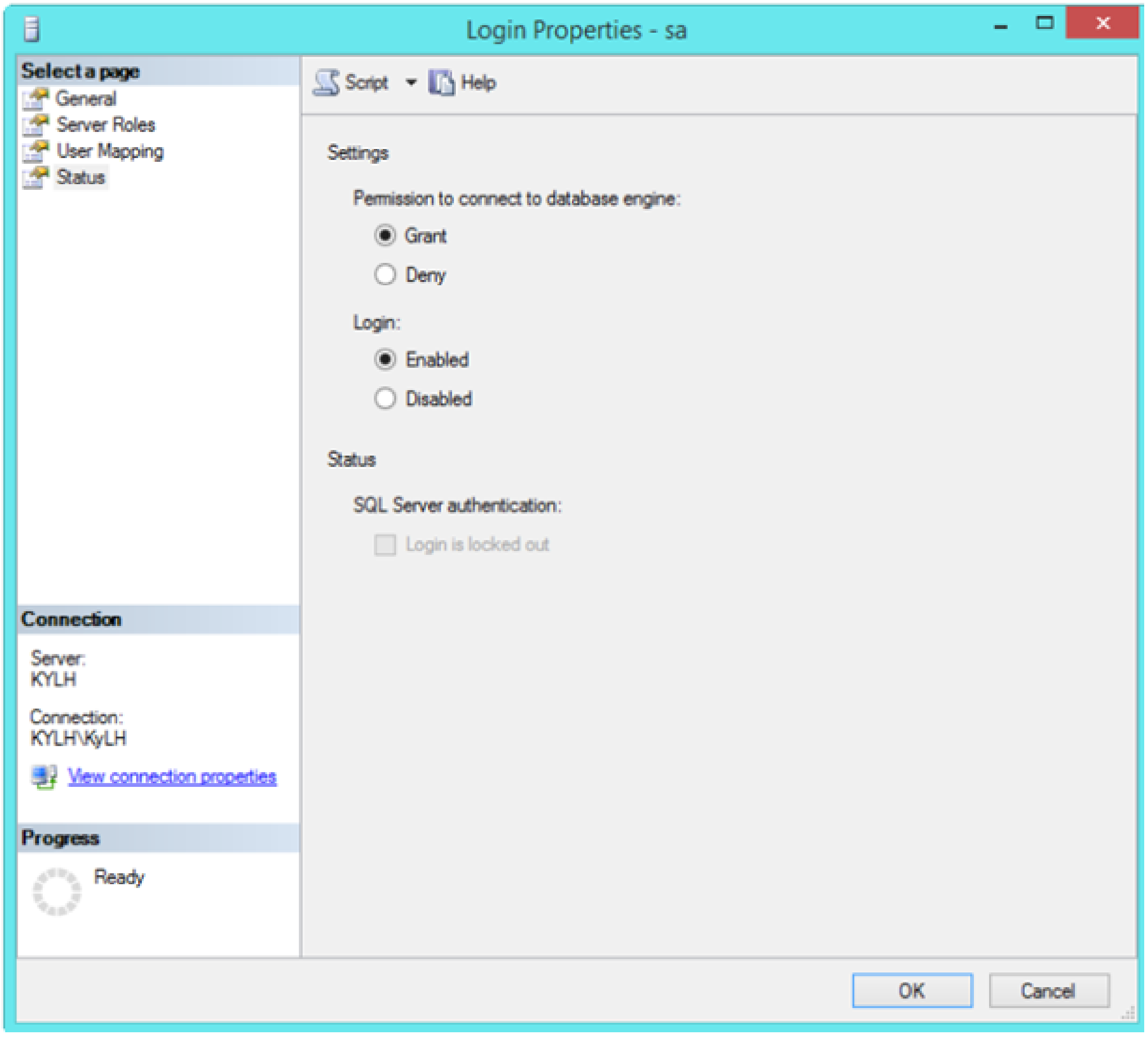
Mapped Credentials

Credential	Provider

Default database: master
Default language: English

Figure 4.5: Setting password for sa

- Enable the sa account by selecting Enable in the login section (by default this account is disabled).



Settings

Permission to connect to database engine:

[✓ Grant] Deny

Login:

[✓ Enabled] Disabled

Status

SQL Server authentication:

☐ Login is locked out

Server:
KYLH

Connection:
KYLH\KyLH

Ready

Figure 4.6: Enable the sa account

2. Establish connection

- Setup the SQL Server to allow login using SQL Server's account.
    + Right-click **SQL Server**, then select **Properties**.

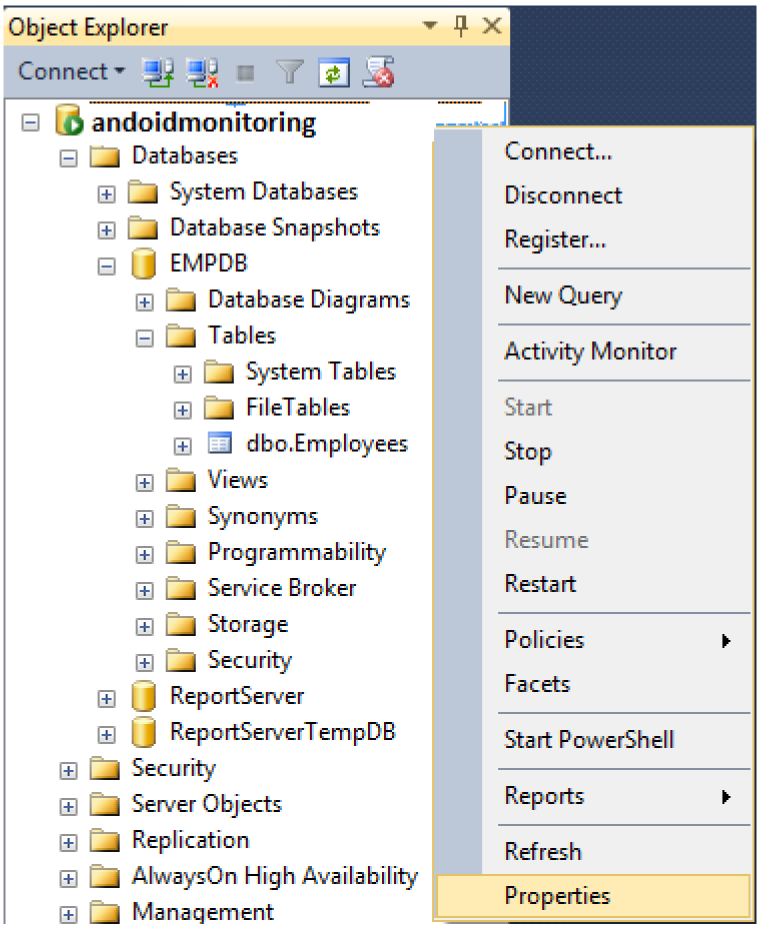
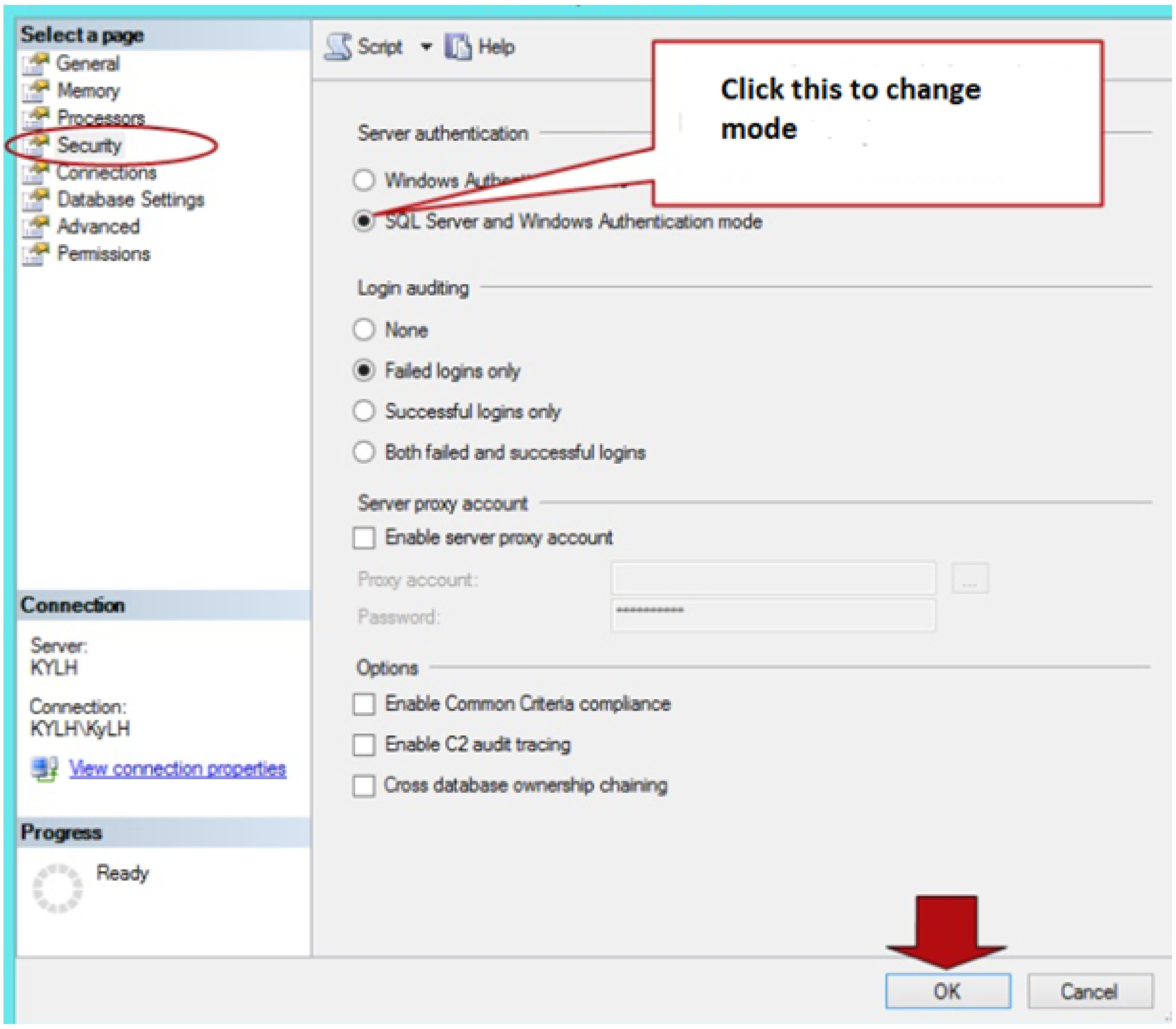


Figure 4.7: Setup the connection

- Select **SQL Server and Windows Authentication** mode.



Connection

Server:
KYLH

Connection:
KYLH\KyLH

View connection properties

Progress

Ready

Server authentication

- ( ) Windows Authen
- (●) SQL Server and Windows Authentication mode

Login auditing

- ( ) None
- (●) Failed logins only
- ( ) Successful logins only
- ( ) Both failed and successful logins

Server proxy account

- [ ] Enable server proxy account

Proxy account:

Password: ***********

Options

- [ ] Enable Common Criteria compliance
- [ ] Enable C2 audit tracing
- [ ] Cross database ownership chaining

Figure 4.8: Change the security mode

- Next, we restart the SQL Server by right-clicking the server name and click **Restart**.

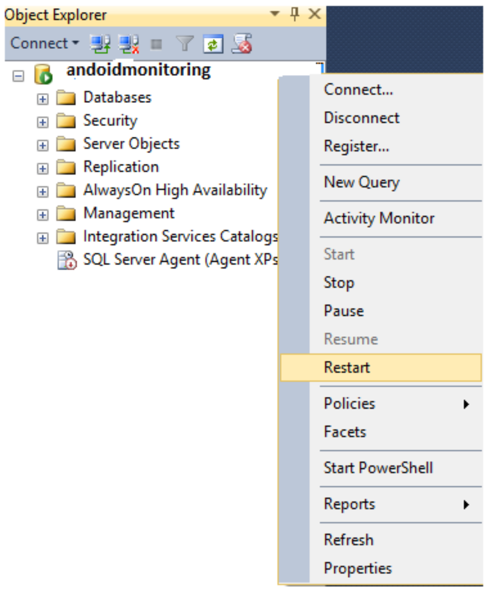


Figure 4.9: Restart the SQL Server

4.2 - Create JPanel Form

- Firstly, Right-click on the **PDMproject** package, select **New**, and then **JFrame Form,** and create a new jFrame Form named **frmMain**

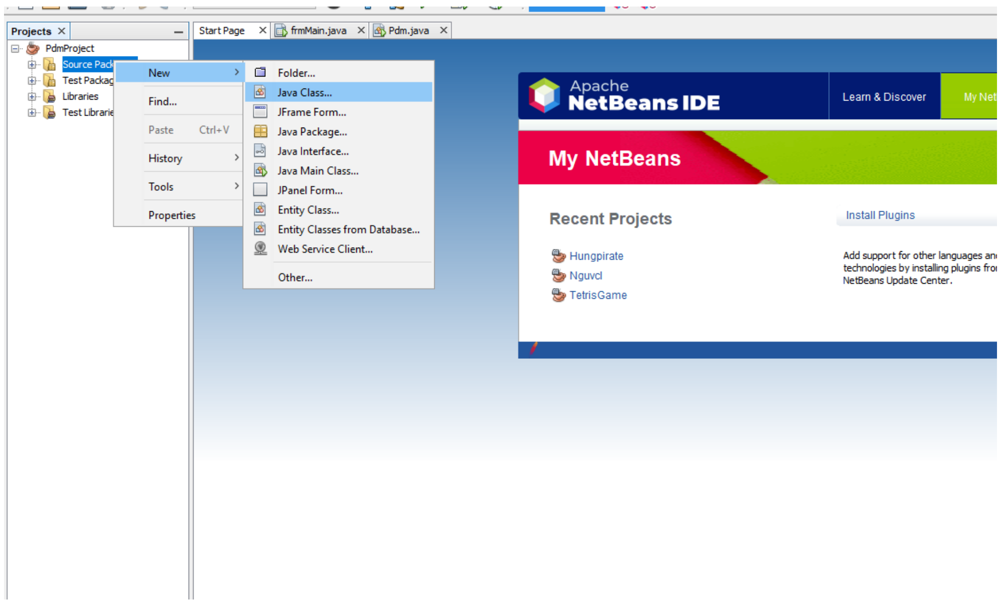


Figure 4.10: Create new class in netbeans

1. **Adding a new library**

+ The tool library we use here is JDBC (l*ink to download:* *[Download - JDBC Driver for SQL Server | Microsoft Docs]*)
+ Based on the version of Netbeans that the computer has pre-installed, you will choose the appropriate JDBC. here Netbean is **JDK 14**, so it will be suitable for **JDBC 8.4**
+ Next, right-click on **Libraries,** select **Add JAR/Folder**:

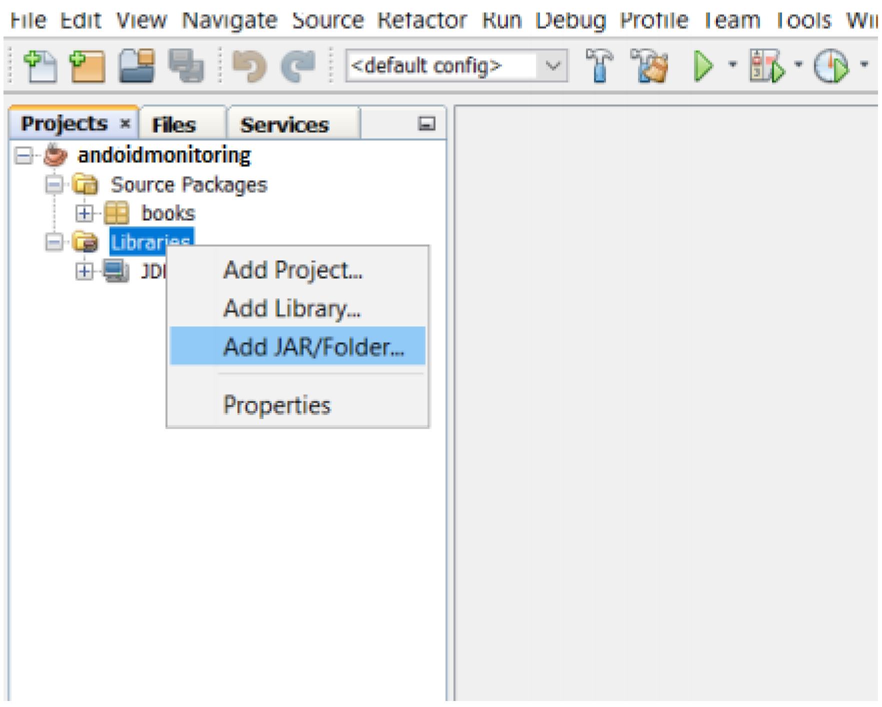
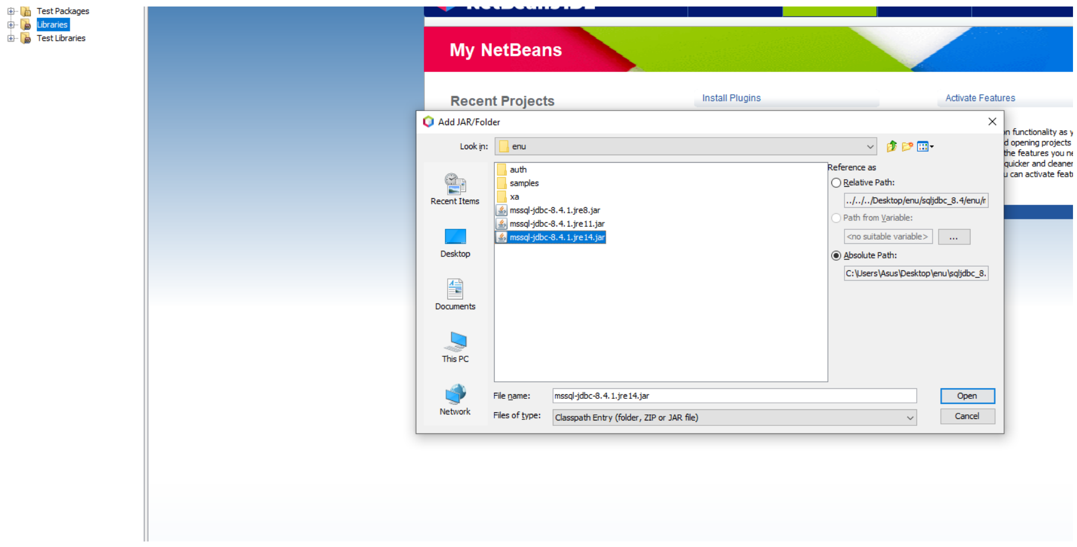


Figure 4.11: Add new libraries

- A new console appearance, select **mssql-jdbc8.4-jre14.jar** and click **open:**


Add JAR/Folder

Look in: enu

auth
samples
xa
mssql-jdbc-8.4.1.jre8.jar
mssql-jdbc-8.4.1.jre11.jar
mssql-jdbc-8.4.1.jre14.jar

Reference as
Relative Path: ../../../Desktop/enu/sqljdbc_8.4/enu/r
Path from Variable: <no suitable variable>
[✓ Absolute Path:] C:\Users\Asus\Desktop\enu\sqljdbc_8.

File name: mssql-jdbc-8.4.1.jre14.jar
Files of type: Classpath Entry (folder, ZIP or JAR file)

Figure 4.12: Use the mssql-jdbc8.4-jre14.jar library

- **Result**: A new library will be added:

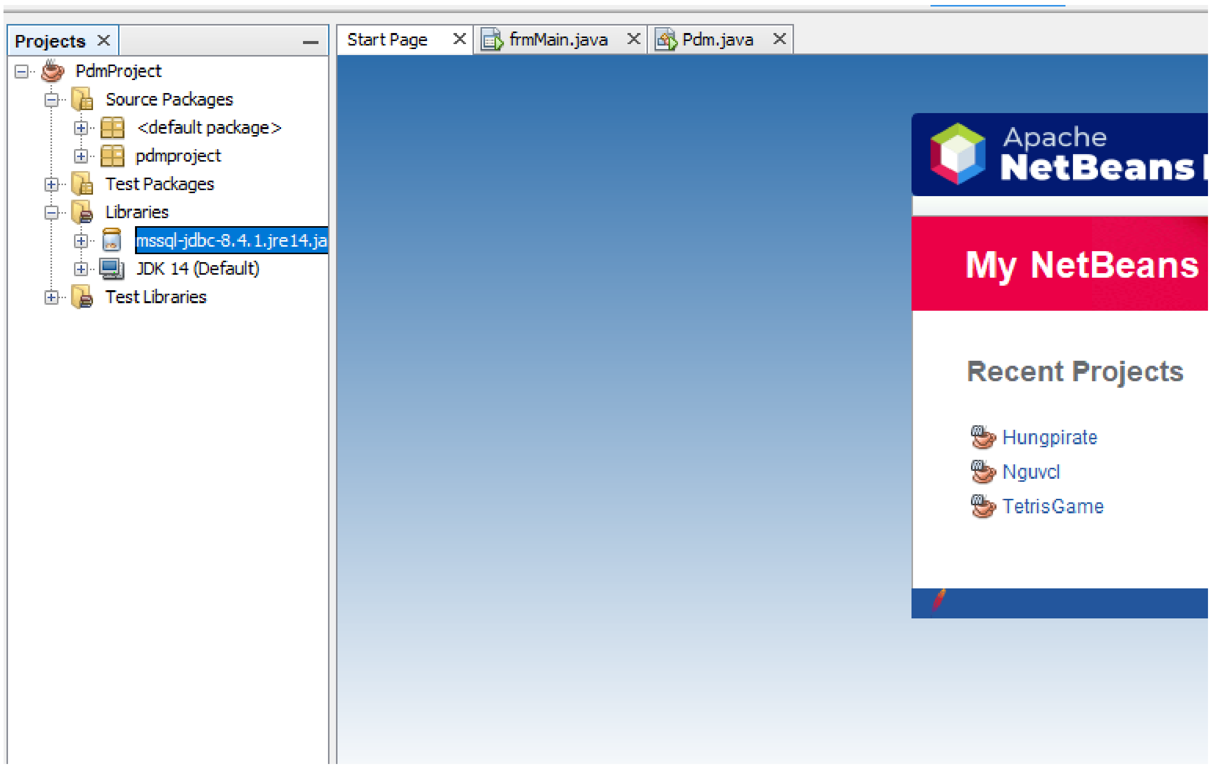
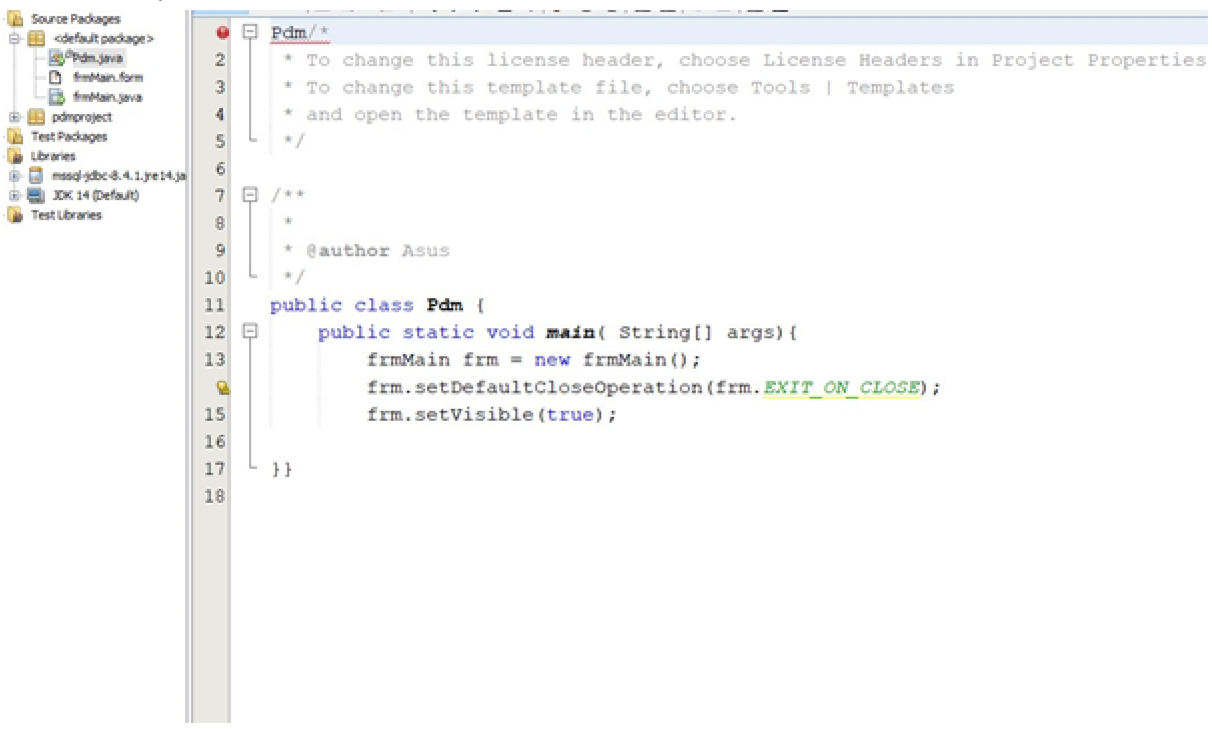


Figure 4.13: Result of netbeans setup

2. **Design a user interface as follows, using Netbean common controls:**

- Select and open the file PDM.java, from the main function, enter code to display the newly created form.



```java
Pdm/*
 * To change this license header, choose License Headers in Project Properties
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Asus
 */
public class Pdm {
    public static void main( String[] args){
        frmMain frm = new frmMain();
        frm.setDefaultCloseOperation(frm.EXIT_ON_CLOSE);
        frm.setVisible(true);

}}
```

Figure 4.14: Create a form by Java

- Run Project Andoirmonitoring and you will see the result like below:

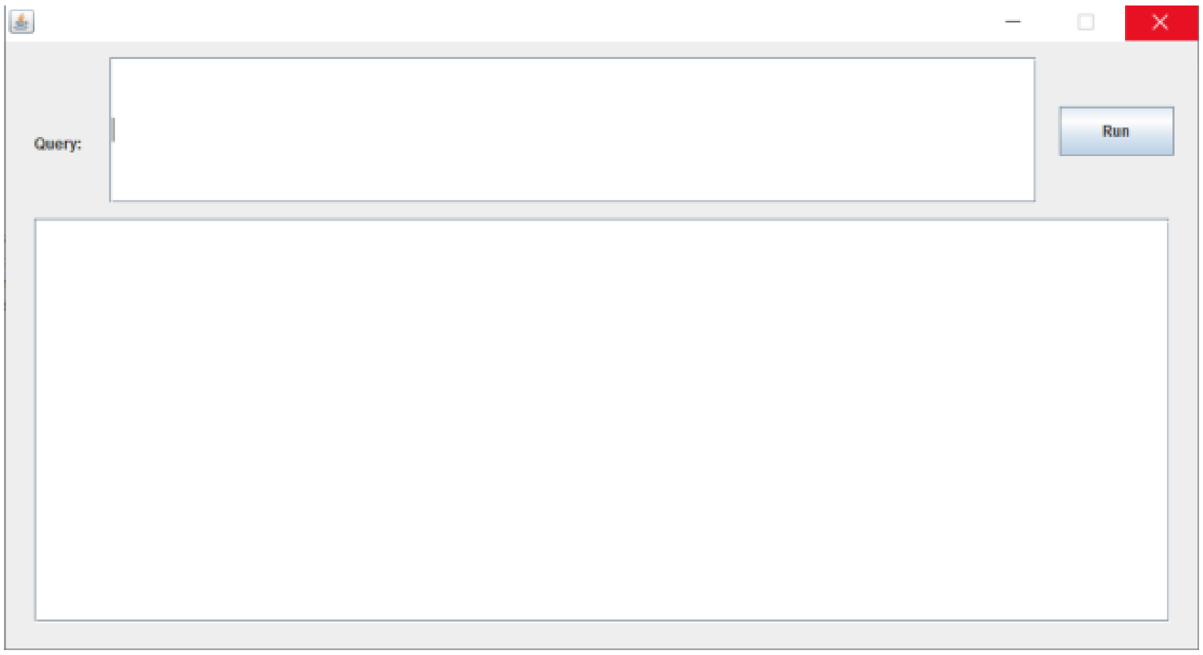
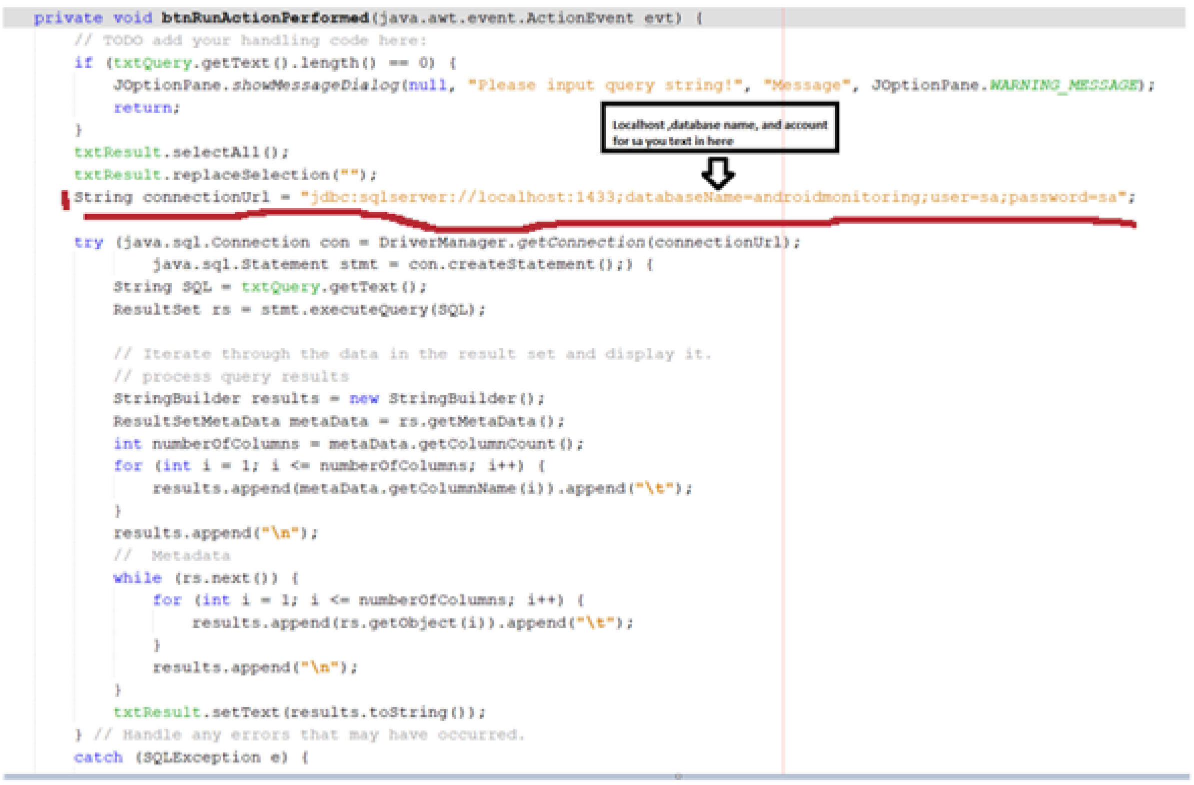


Figure 4.15: Result of the Java program

4.3 - Connect Java And Database

- Open source code of **frmMain.java** and copy the code from this link: [File code - Pastebin.com]()
- In String **connectionUrl** is have a form:
    + *jdbc:sqlserver://[serverName[\instanceName][:portNumber]][; property = value [;   property = value]]*

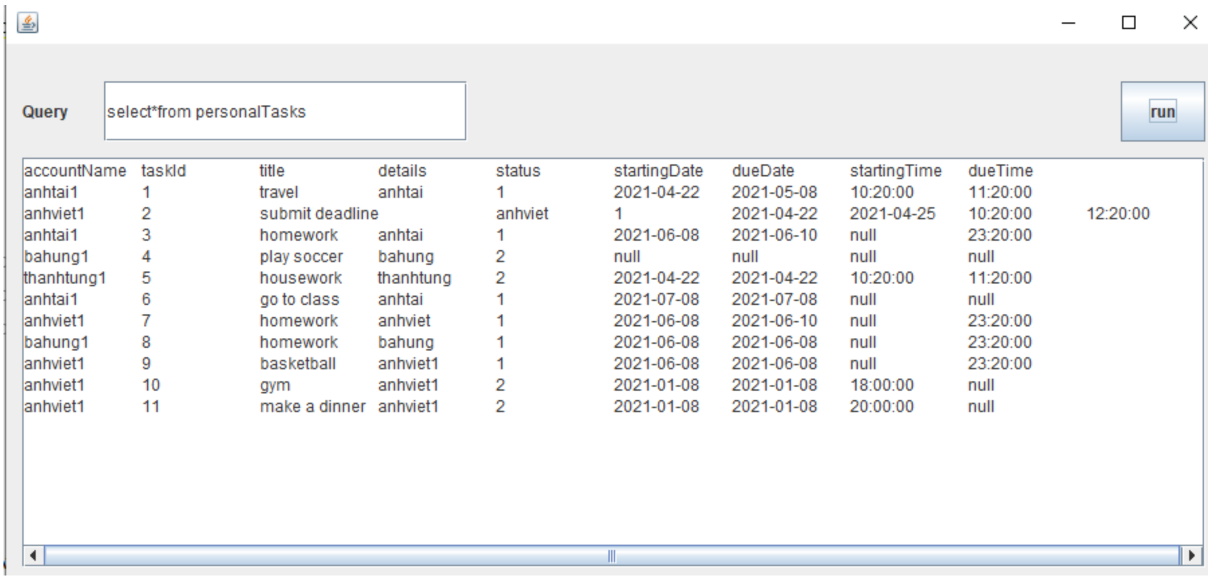


```java
private void btnRunActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (txtQuery.getText().length() == 0) {
        JOptionPane.showMessageDialog(null, "Please input query string!", "Message", JOptionPane.WARNING_MESSAGE);
        return;
    }
    txtResult.selectAll();
    txtResult.replaceSelection("");
    String connectionUrl = "jdbc:sqlserver://localhost:1433;databaseName=androidmonitoring;user=sa;password=sa";

    try (java.sql.Connection con = DriverManager.getConnection(connectionUrl);
            java.sql.Statement stmt = con.createStatement();) {
        String SQL = txtQuery.getText();
        ResultSet rs = stmt.executeQuery(SQL);

        // Iterate through the data in the result set and display it.
        // process query results
        StringBuilder results = new StringBuilder();
        ResultSetMetaData metaData = rs.getMetaData();
        int numberOfColumns = metaData.getColumnCount();
        for (int i = 1; i <= numberOfColumns; i++) {
            results.append(metaData.getColumnName(i)).append("\t");
        }
        results.append("\n");
        //  Metadata
        while (rs.next()) {
            for (int i = 1; i <= numberOfColumns; i++) {
                results.append(rs.getObject(i)).append("\t");
            }
            results.append("\n");
        }
        txtResult.setText(results.toString());
    } // Handle any errors that may have occurred.
    catch (SQLException e) {
```

Figure 4.16: Change the connectionUrl

Final result:

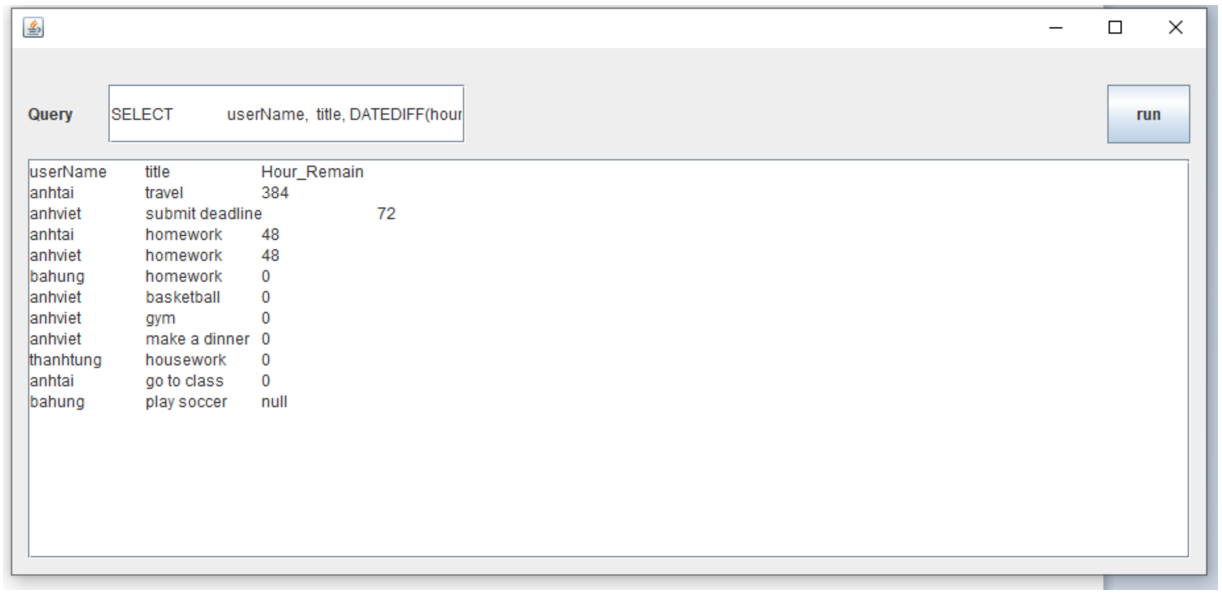
- Users can interact with the database by query in the **Query area** and lick **Run button**:



```
select*from personalTasks
```

```
accountName  taskId   title           details    status    startingDate  dueDate     startingTime  dueTime
anhtai1      1        travel          anhtai     1         2021-04-22    2021-05-08  10:20:00      11:20:00
anhviet1     2        submit deadline            anhviet   1             2021-04-22  2021-04-25    10:20:00   12:20:00
anhtai1      3        homework        anhtai     1         2021-06-08    2021-06-10  null          23:20:00
bahung1      4        play soccer     bahung     2         null          null        null          null
thanhtung1   5        housework       thanhtung  2         2021-04-22    2021-04-22  10:20:00      11:20:00
anhtai1      6        go to class     anhtai     1         2021-07-08    2021-07-08  null          null
anhviet1     7        homework        anhviet    1         2021-06-08    2021-06-10  null          23:20:00
bahung1      8        homework        bahung     1         2021-06-08    2021-06-08  null          23:20:00
anhviet1     9        basketball      anhviet1   1         2021-06-08    2021-06-08  null          23:20:00
anhviet1     10       gym             anhviet1   2         2021-01-08    2021-01-08  18:00:00      null
anhviet1     11       make a dinner   anhviet1   2         2021-01-08    2021-01-08  20:00:00      null
```

Figure 4.17: Result of the first testing query



```
SELECT          userName,  title, DATEDIFF(hour
```

```
userName     title          Hour_Remain
anhtai       travel         384
anhviet      submit deadline              72
anhtai       homework       48
anhviet      homework       48
bahung       homework       0
anhviet      basketball     0
anhviet      gym            0
anhviet      make a dinner  0
thanhtung    housework      0
anhtai       go to class    0
bahung       play soccer    null
```

Figure 4.18: Result of the second testing query

CHAPTER 5: APPLICATION IN REAL LIFE

5.1 - Overview

______Android Task Monitoring is a mobile app made by React Native - a powerful Facebook framework. MySQL is used to create a database to support my application. We tried our best to make the application satisfy many strict requirements of an application in real life which include UI/UX, security, and connecting smoothly between frontend and backend,... . However, our project is still in the development stage so it only works on localhost. The list of requirements of the project is mentioned in **section 2.1**. Despite the project's name, Android Task Monitoring is developed by React Native - a cross-platform, so IOS can use this app too.

5.2 - Analysis

- Our app will work based on the diagram below:

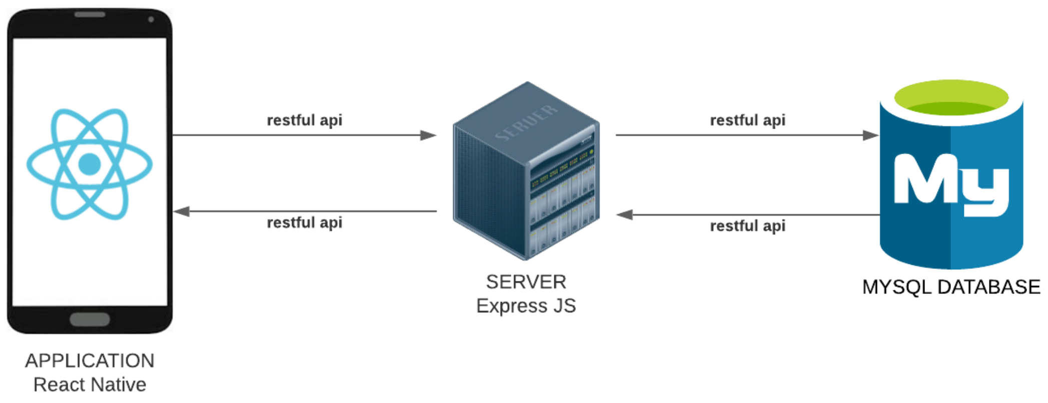
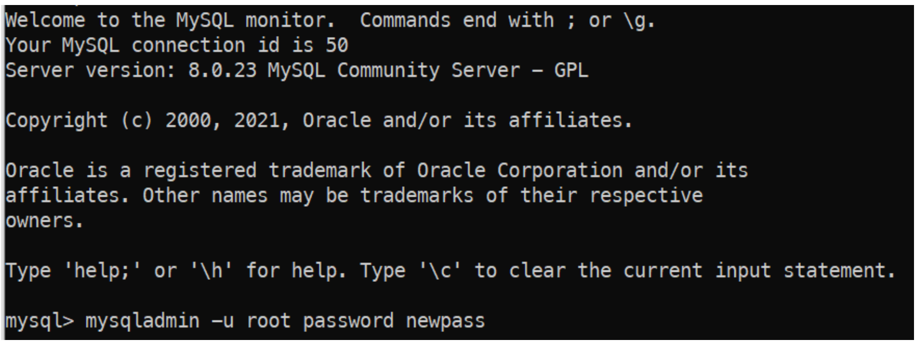


Figure 5.1: Diagram about application working process

- **Backend:** We will use Express JS - a Node js application framework to create a server to connect with React Native application and MySQL database. To connect with the database, our server uses the MySQL2 library to initialize the query which we want to get data from the database.
- **Frontend:** React native with some additional libraries are used to design and implement many functions which we develop for Android Tasking Monitoring. Based on the requirements, we divided our application into 5 main screens. These are:
  + Sign-up Screen: For users to create a new account with a username and full name and a password.
  + Sign-in Screen: This screen is a login form for user sign-in.
  + Personal Screen: The list of personal tasks will show here
  + Team Screen: In the team screen, users can manage all teams they are participating.
  + Info Screen: This screen shows the information of the user who is logging in and a button for the user log out.

5.3 - MySQL Database

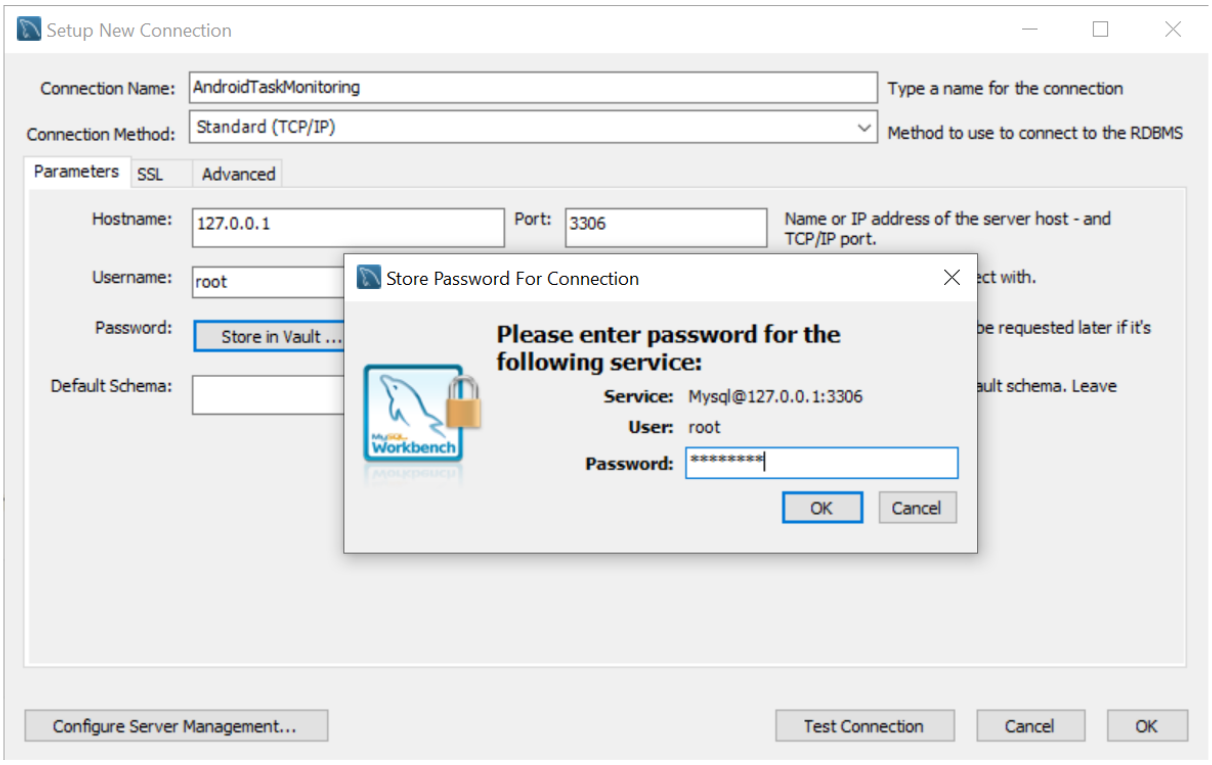
- Firstly, we create a password for the root user to connect. In this case, I set up the username as root and the password as 'password'. From the server side, we must configure for the connection is root and password before initializing any query to the database if we do not want our connection to be broken.
    + In case this is the first time you set up a new password, run the command in the mysql command line: *mysqladmin -u root password newpass* with newpass is the password you want.



```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 50
Server version: 8.0.23 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> mysqladmin -u root password newpass
```

Figure 5.2: Set a password for a new MySQL account

    + If you have a password before, you can use this for each connection or change to a new password by the command: *mysqladmin -u root -p password newpass*
- Next step, in the MySQL Workbench, you must create a new connection with the username is root and password created from the previous step. Click OK. We need to do this thing because our database and its schema need to be initialized first.



Setup New Connection

Connection Name: AndroidTaskMonitoring — Type a name for the connection

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Parameters | SSL | Advanced

Hostname: 127.0.0.1 Port: 3306 — Name or IP address of the server host - and TCP/IP port.

Username: root — ct with.

Password: Store in Vault ... — be requested later if it's

Default Schema: — ult schema. Leave

Store Password For Connection

Please enter password for the following service:

Service: Mysql@127.0.0.1:3306

User: root

Password: ********

OK | Cancel

Configure Server Management... | Test Connection | Cancel | OK

Figure 5.3: Create a connection to MySQL database

- In the query screen, we will create our database and the schema of our database which is mentioned in **section 2.3**. These are all the queries to create our database schema:

```
-- Create the database
CREATE DATABASE AndroidTaskMonitoring;
USE AndroidTaskMonitoring; -- use database to create schema table
-- Create Users-schema
CREATE TABLE Users (
        username varchar(255) PRIMARY KEY,
        password varchar(255) NOT NULL,
        fullname varchar(255) NOT NULL
);
-- Create PersonalTasks-schema
CREATE TABLE PersonalTasks (
        PTID int PRIMARY KEY,
        owner varchar(255) NOT NULL,
        title varchar(255) NOT NULL,
        details varchar(255),
        status int NOT NULL,
        startDate date NOT NULL,
        startTime time NOT NULL,
        dueDate date NOT NULL,
        dueTime time NOT NULL,
        FOREIGN KEY (owner) REFERENCES Users(username)
);
-- Create Teams-schema
CREATE TABLE Teams (
        TID int PRIMARY KEY,
        name varchar(255) NOT NULL,
        details varchar(255),
        manager varchar(255) NOT NULL,
        FOREIGN KEY (manager) REFERENCES Users(username)
);
-- Create Employees-schema
CREATE TABLE Employees (
        username varchar(255),
        TID int,
        PRIMARY KEY (username, TID),
        FOREIGN KEY (username) REFERENCES Users(username),
        FOREIGN KEY (TID) REFERENCES Teams(TID) ON DELETE CASCADE
);
-- Create TeamTasks-schema
CREATE TABLE TeamTasks (
        TTID int PRIMARY KEY,
        TID int NOT NULL,
        title varchar(255) NOT NULL,
        details varchar(255),
        status int NOT NULL,
        startDate date NOT NULL,
        startTime time NOT NULL,
        dueDate date NOT NULL,
        dueTime time NOT NULL,
        FOREIGN KEY (TID) REFERENCES Teams(TID) ON DELETE CASCADE
);
-- Create Participants-schema
CREATE TABLE Participants (
        username varchar(255),
        TTID int,
        PRIMARY KEY (username, TTID),
        FOREIGN KEY (username) REFERENCES Users(username),
        FOREIGN KEY (TTID) REFERENCES TeamTasks(TTID) ON DELETE
CASCADE
);
```

- When you click Execute, this should not have any error messages. You are done preparing the database to connect with the application.

5.4 - Android Task Monitoring By React Native

______In this section, we will talk about how we created a real application step by step. Many things must be installed and some libraries may conflict. However, we will try our best to describe our way to accomplish this project. You can find more information and our source code from GitHub: https://github.com/vietanhvo/Android_Task_Monitoring (this repository is the source code of the full application) and https://github.com/vietanhvo/server_ATM (this is the backend for the project). We will talk about two parts: backend and frontend in order.

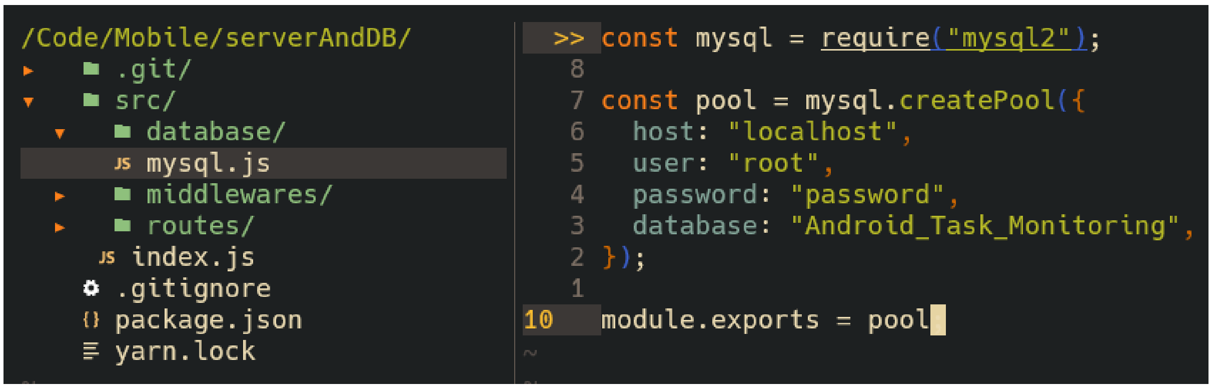
1. **Backend**

- Set up:

Express JS will be used as the framework for us to create the backend. The instruction to install it is in this link https://expressjs.com/en/starter/installing.html. Next step, we find a library to support our server to connect with the MySQL database. MySQL2 (https://github.com/sidorares/node-mysql2) which you can install through npm or yarn is the library that supports our needs.

- Configure connection:

Firstly, we need to create a mysql.js file to configure the connection and our database. We will use the database name "Android_Task_Monitoring" with username as "root" and password as "password". Because we only run this on localhost, localhost is the destination link to connect to the MySQL server.



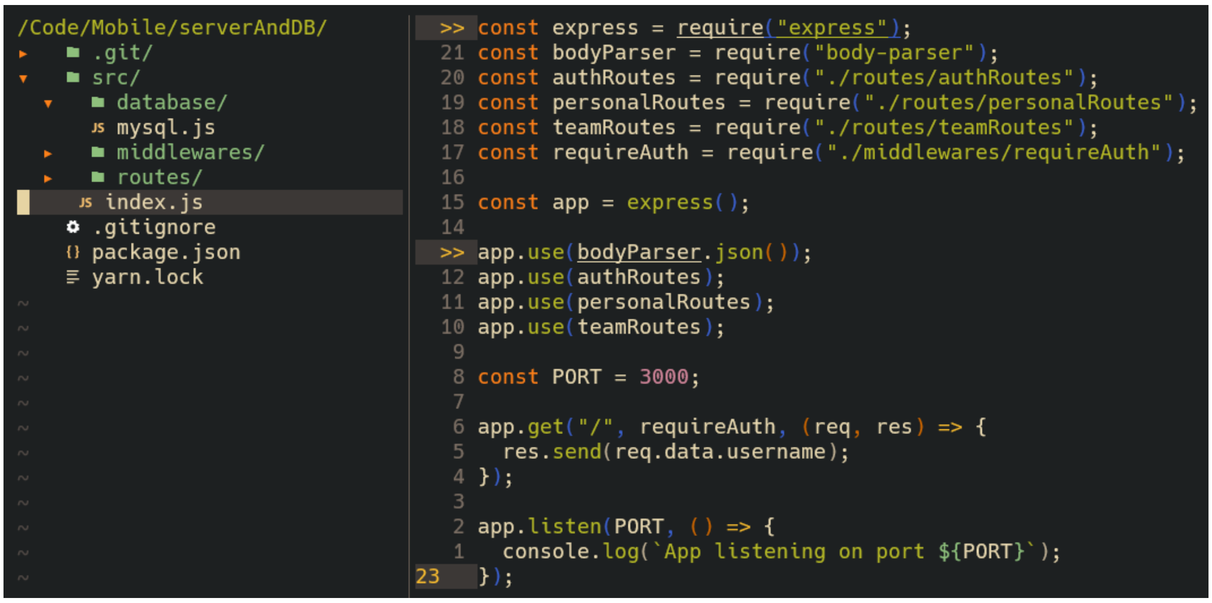
```
/Code/Mobile/serverAndDB/
▸ .git/
▾ src/
  ▾ database/
      mysql.js
  ▸ middlewares/
  ▸ routes/
    index.js
  .gitignore
  package.json
  yarn.lock
```

```js
const mysql = require("mysql2");

const pool = mysql.createPool({
  host: "localhost",
  user: "root",
  password: "password",
  database: "Android_Task_Monitoring",
});

module.exports = pool
```

Figure 5.4: Setup database connection properties

- Backend code

  + index.js: this is the main code of our server. Whenever we run the server, it will initialize this file. The idea is we will open a port 3000 on my computer for the server to run in this port. The details about how to connect are in files in the routes folder.

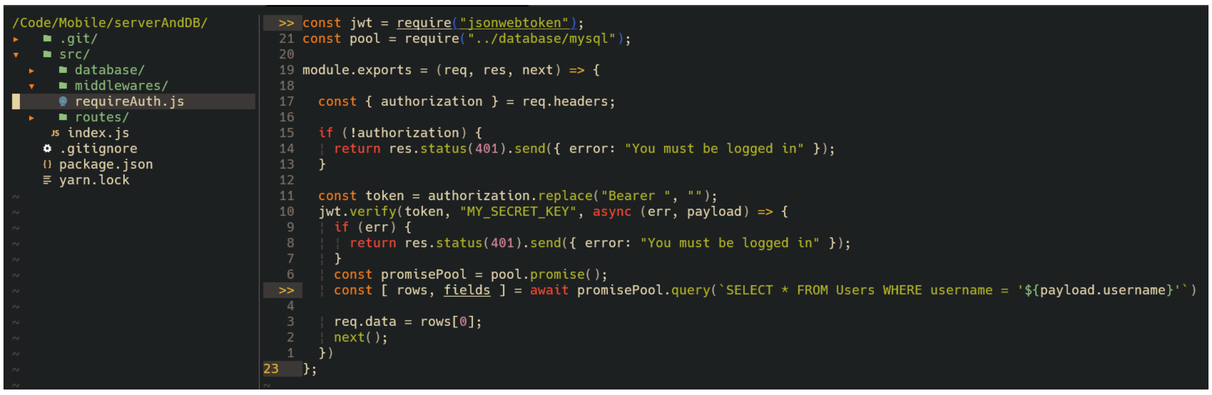


```
/Code/Mobile/serverAndDB/
▸ .git/
▾ src/
  ▾ database/
      mysql.js
  ▸ middlewares/
  ▸ routes/
    index.js
  .gitignore
  package.json
  yarn.lock
```

```js
const express = require("express");
const bodyParser = require("body-parser");
const authRoutes = require("./routes/authRoutes");
const personalRoutes = require("./routes/personalRoutes");
const teamRoutes = require("./routes/teamRoutes");
const requireAuth = require("./middlewares/requireAuth");

const app = express();

app.use(bodyParser.json());
app.use(authRoutes);
app.use(personalRoutes);
app.use(teamRoutes);

const PORT = 3000;

app.get("/", requireAuth, (req, res) => {
  res.send(req.data.username);
});

app.listen(PORT, () => {
  console.log(`App listening on port ${PORT}`);
});
```

Figure 5.5: Main code of Express server

  + middlewares: this folder is used for authentication progress by JWT standard. We use json web token (JWT) - a standard for securely transmitting information between parties as a JSON object. The idea is whenever a request comes to our server, this must be sent with the header is the JWT. This token is checked valid or not by a secret key only the server has. If not valid, users must log in again.



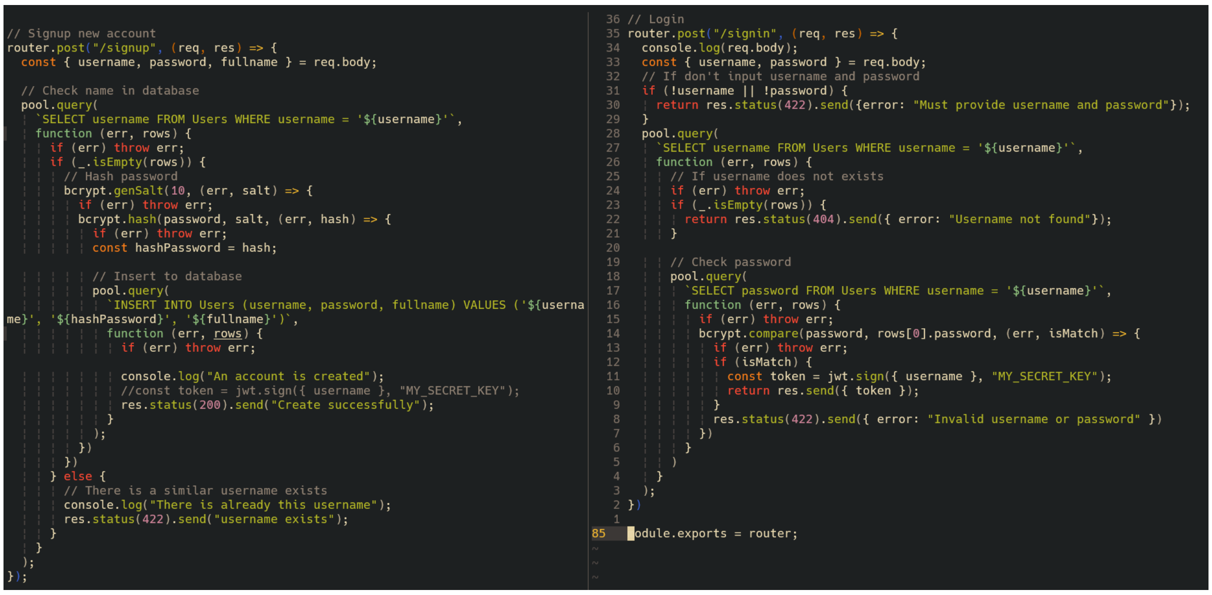
```
/Code/Mobile/serverAndDB/
▸ .git/
▾ src/
  ▸ database/
  ▾ middlewares/
      requireAuth.js
  ▸ routes/
    index.js
  .gitignore
  package.json
  yarn.lock
```

```javascript
const jwt = require("jsonwebtoken");
const pool = require("../database/mysql");

module.exports = (req, res, next) => {

  const { authorization } = req.headers;

  if (!authorization) {
    return res.status(401).send({ error: "You must be logged in" });
  }

  const token = authorization.replace("Bearer ", "");
  jwt.verify(token, "MY_SECRET_KEY", async (err, payload) => {
    if (err) {
      return res.status(401).send({ error: "You must be logged in" });
    }
    const promisePool = pool.promise();
    const [ rows, fields ] = await promisePool.query(`SELECT * FROM Users WHERE username = '${payload.username}'`)

    req.data = rows[0];
    next();
  })
};
```

Figure 5.6: Authenticate middleware for server

+ routes: this folder is used for actions that will do for each connection.

1. **authRoutes.js:** This file is used for register and login actions.
   When users request to "/signup", they must pass the username, password, and full name in the body. The server will check if this username has already existed in the database or not. If this username existed, users must choose another username. If everything is valid, their information will be inserted into the database (Users table). At this step, we will hash the password to save the user's information more securely before inserting it into the database.
   When users request to "/signin", they must pass the username, and password in the body. When a user logs in, our server will connect to the database and check if this username exists and password is valid or not. If everything is correct, JWT will be created by the server and respond to a user who logs in successfully. From the user side, the device must store this token and use it for the next request. If the username or password is wrong, they must be logged in again with different information.

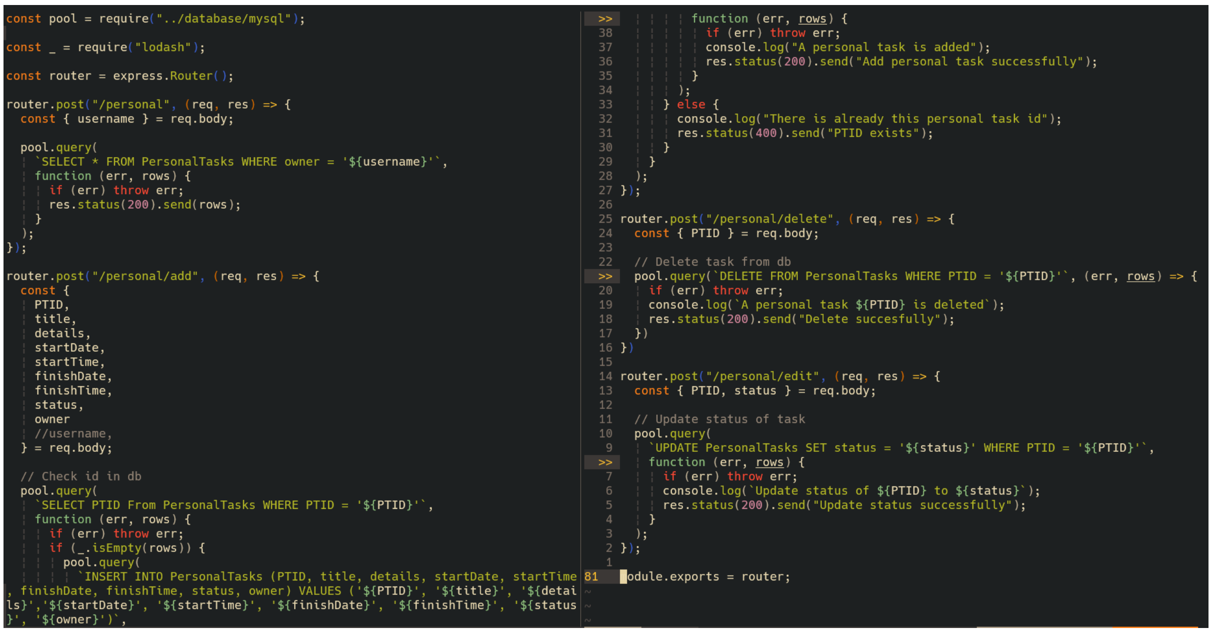


```javascript
// Signup new account
router.post("/signup", (req, res) => {
  const { username, password, fullname } = req.body;

  // Check name in database
  pool.query(
    `SELECT username FROM Users WHERE username = '${username}'`,
    function (err, rows) {
      if (err) throw err;
      if (_.isEmpty(rows)) {
        // Hash password
        bcrypt.genSalt(10, (err, salt) => {
          if (err) throw err;
          bcrypt.hash(password, salt, (err, hash) => {
            if (err) throw err;
            const hashPassword = hash;

            // Insert to database
            pool.query(
              `INSERT INTO Users (username, password, fullname) VALUES ('${userna
me}', '${hashPassword}', '${fullname}')`,
              function (err, rows) {
                if (err) throw err;

                console.log("An account is created");
                //const token = jwt.sign({ username }, "MY_SECRET_KEY");
                res.status(200).send("Create successfully");
              }
            );
          })
        })
      } else {
        // There is a similar username exists
        console.log("There is already this username");
        res.status(422).send("username exists");
      }
    }
  );
});
```

```javascript
// Login
router.post("/signin", (req, res) => {
  console.log(req.body);
  const { username, password } = req.body;
  // If don't input username and password
  if (!username || !password) {
    return res.status(422).send({error: "Must provide username and password"});
  }
  pool.query(
    `SELECT username FROM Users WHERE username = '${username}'`,
    function (err, rows) {
      // If username does not exists
      if (err) throw err;
      if (_.isEmpty(rows)) {
        return res.status(404).send({ error: "Username not found"});
      }

      // Check password
      pool.query(
        `SELECT password FROM Users WHERE username = '${username}'`,
        function (err, rows) {
          if (err) throw err;
          bcrypt.compare(password, rows[0].password, (err, isMatch) => {
            if (err) throw err;
            if (isMatch) {
              const token = jwt.sign({ username }, "MY_SECRET_KEY");
              return res.send({ token });
            }
            res.status(422).send({ error: "Invalid username or password" })
          })
        }
      )
    }
  );
})

module.exports = router;
```

Figure 5.7: Authenticate flow for the app with login and signup

2. **personalRoutes.js:** this file is used for personal task actions.
   Personal tasks have abilities to add tasks, delete tasks, update status, and load tasks from the database. This the reason why we divide them into each route:
   - "/personal": when there is a request posted to this route, that means users want to load tasks to his/her device. The server will connect to the database and query "SELECT * FROM PersonalTasks WHERE owner = 'username'". Then, it will respond to users.
   - "/personal/add": when there is a request posted to this route, that means users want to add new tasks to the database. The server will receive information and add them to the database.
   - "/personal/delete": when there is a request posted to this route, that means users want to delete a task from the database. They must provide the task TTID to specify which task they want to delete. The query "DELETE FROM PersonalTasks WHERE PTID = 'PTID-provided'" will be initialized.
   - "/personal/edit": when there is a request posted to this route, that means users want to update that they have done the task or are doing it. From the database, by convention 1 is done and 0 is in progress. The server will do a query "UPDATE PersonalTasks SET STATUS = '0/1' WHERE PTID = 'PTID-provided'" to the database.



```javascript
const pool = require("../database/mysql");

const _ = require("lodash");

const router = express.Router();

router.post("/personal", (req, res) => {
  const { username } = req.body;

  pool.query(
    `SELECT * FROM PersonalTasks WHERE owner = '${username}'`,
    function (err, rows) {
      if (err) throw err;
      res.status(200).send(rows);
    }
  );
});

router.post("/personal/add", (req, res) => {
  const {
    PTID,
    title,
    details,
    startDate,
    startTime,
    finishDate,
    finishTime,
    status,
    owner
    //username,
  } = req.body;

  // Check id in db
  pool.query(
    `SELECT PTID From PersonalTasks WHERE PTID = '${PTID}'`,
    function (err, rows) {
      if (err) throw err;
      if (_.isEmpty(rows)) {
        pool.query(
          `INSERT INTO PersonalTasks (PTID, title, details, startDate, startTime
, finishDate, finishTime, status, owner) VALUES ('${PTID}', '${title}', '${detai
ls}','${startDate}', '${startTime}', '${finishDate}', '${finishTime}', '${status
}', '${owner}')`,
```

```javascript
          function (err, rows) {
            if (err) throw err;
            console.log("A personal task is added");
            res.status(200).send("Add personal task successfully");
          }
        );
      } else {
        console.log("There is already this personal task id");
        res.status(400).send("PTID exists");
      }
    }
  );
});

router.post("/personal/delete", (req, res) => {
  const { PTID } = req.body;

  // Delete task from db
  pool.query(`DELETE FROM PersonalTasks WHERE PTID = '${PTID}'`, (err, rows) => {
    if (err) throw err;
    console.log(`A personal task ${PTID} is deleted`);
    res.status(200).send("Delete succesfully");
  })
})

router.post("/personal/edit", (req, res) => {
  const { PTID, status } = req.body;

  // Update status of task
  pool.query(
    `UPDATE PersonalTasks SET status = '${status}' WHERE PTID = '${PTID}'`,
    function (err, rows) {
      if (err) throw err;
      console.log(`Update status of ${PTID} to ${status}`);
      res.status(200).send("Update status successfully");
    }
  );
});

module.exports = router;
```

Figure 5.8: Server handles PersonalTasks request

3. **teamRoutes.js:** This file is used for team task actions.

In TeamScreen, users have the ability to create new teams, add new members to a team, add new tasks for their team, update status for tasks, and delete tasks. In fact, all the actions have some similar points to the personal actions but it is more complicated. Because this part included too many things, We only introduced them in general. We must follow some rules such as:

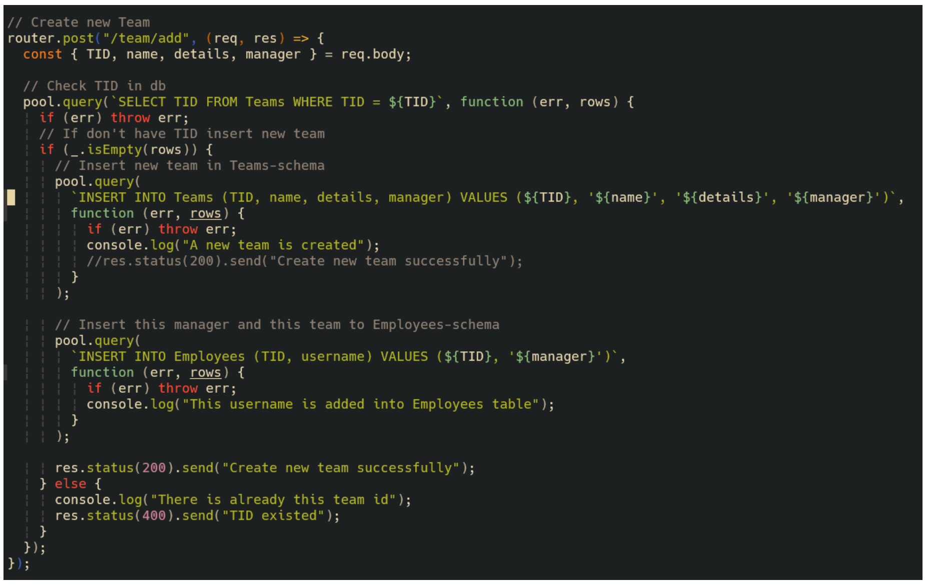
+ In a team, the manager is the only person who has abilities to add new members, and delete tasks.

+ Users when creating a new task must allocate it to at least a person in a team.

+ When looking at tasks, the manager is the only person who can look at all the tasks of all members of a team. Members in a team only see the list of tasks which is allocated for themselves.

- Based on all requirements, we create 9 routes for each action:

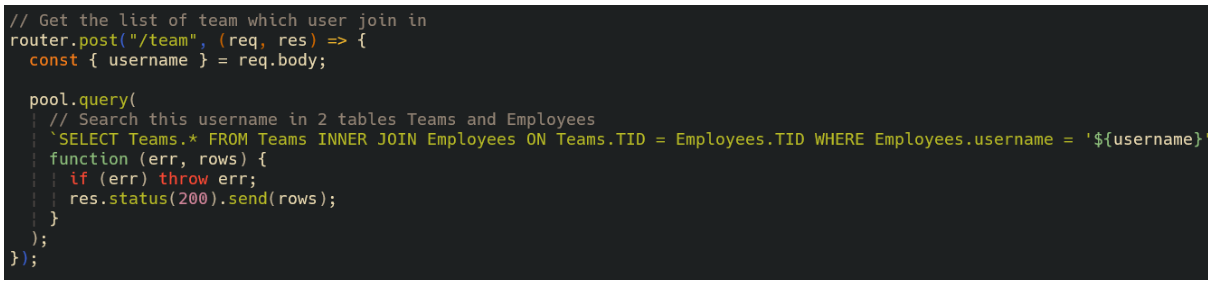
+ “/team/add”: This route creates a new team in the database.



```
// Create new Team
router.post("/team/add", (req, res) => {
  const { TID, name, details, manager } = req.body;

  // Check TID in db
  pool.query(`SELECT TID FROM Teams WHERE TID = ${TID}`, function (err, rows) {
    if (err) throw err;
    // If don't have TID insert new team
    if (_.isEmpty(rows)) {
      // Insert new team in Teams-schema
      pool.query(
        `INSERT INTO Teams (TID, name, details, manager) VALUES (${TID}, '${name}', '${details}', '${manager}')`,
        function (err, rows) {
          if (err) throw err;
          console.log("A new team is created");
          //res.status(200).send("Create new team successfully");
        }
      );

      // Insert this manager and this team to Employees-schema
      pool.query(
        `INSERT INTO Employees (TID, username) VALUES (${TID}, '${manager}')`,
        function (err, rows) {
          if (err) throw err;
          console.log("This username is added into Employees table");
        }
      );

      res.status(200).send("Create new team successfully");
    } else {
      console.log("There is already this team id");
      res.status(400).send("TID existed");
    }
  });
});
```

Figure 5.9: Create a new team

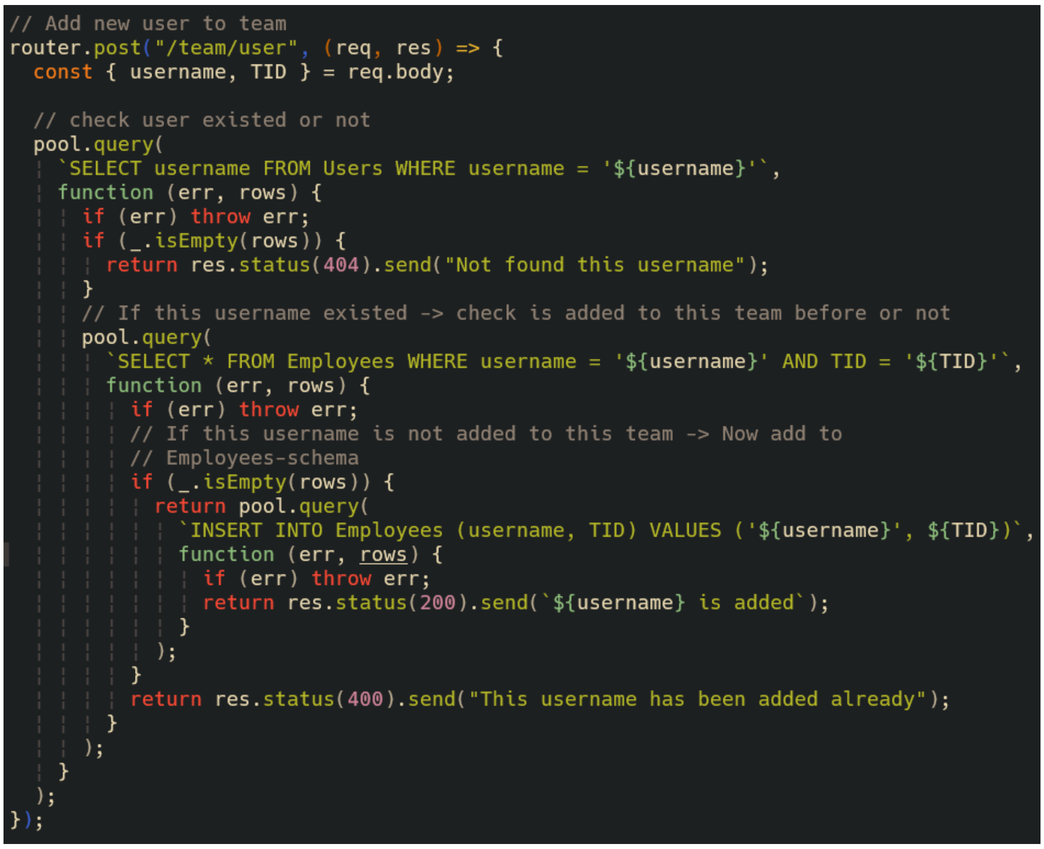
+ “/team”: get the list of team information that a user joins.



```
// Get the list of team which user join in
router.post("/team", (req, res) => {
  const { username } = req.body;

  pool.query(
    // Search this username in 2 tables Teams and Employees
    `SELECT Teams.* FROM Teams INNER JOIN Employees ON Teams.TID = Employees.TID WHERE Employees.username = '${username}'
    function (err, rows) {
      if (err) throw err;
      res.status(200).send(rows);
    }
  );
});
```

Figure 5.10: Get the list of team user joins in

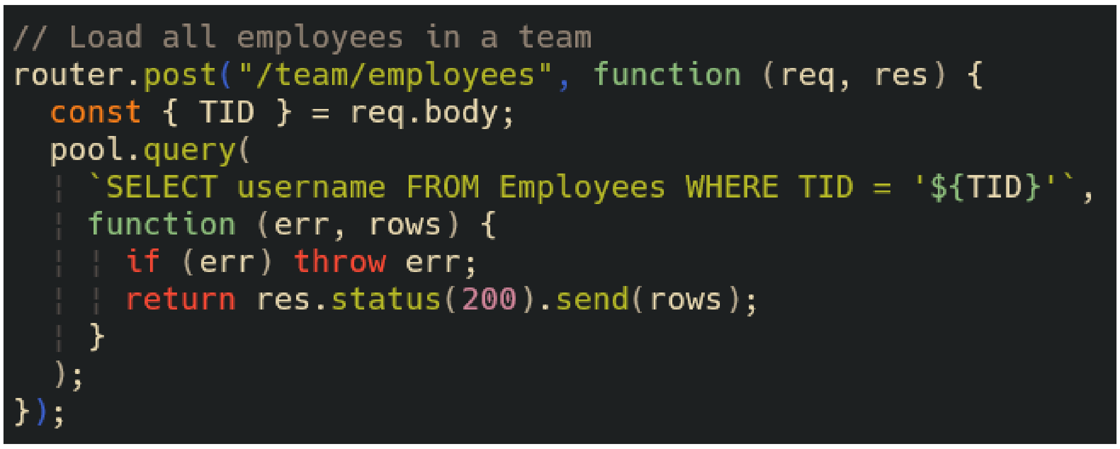
+ “/team/user”: add new members to a team.



```
// Add new user to team
router.post("/team/user", (req, res) => {
  const { username, TID } = req.body;

  // check user existed or not
  pool.query(
    `SELECT username FROM Users WHERE username = '${username}'`,
    function (err, rows) {
      if (err) throw err;
      if (_.isEmpty(rows)) {
        return res.status(404).send("Not found this username");
      }
      // If this username existed -> check is added to this team before or not
      pool.query(
        `SELECT * FROM Employees WHERE username = '${username}' AND TID = '${TID}'`,
        function (err, rows) {
          if (err) throw err;
          // If this username is not added to this team -> Now add to
          // Employees-schema
          if (_.isEmpty(rows)) {
            return pool.query(
              `INSERT INTO Employees (username, TID) VALUES ('${username}', ${TID})`,
              function (err, rows) {
                if (err) throw err;
                return res.status(200).send(`${username} is added`);
              }
            );
          }
          return res.status(400).send("This username has been added already");
        }
      );
    }
  );
});
```

Figure 5.11: Add new users to a team

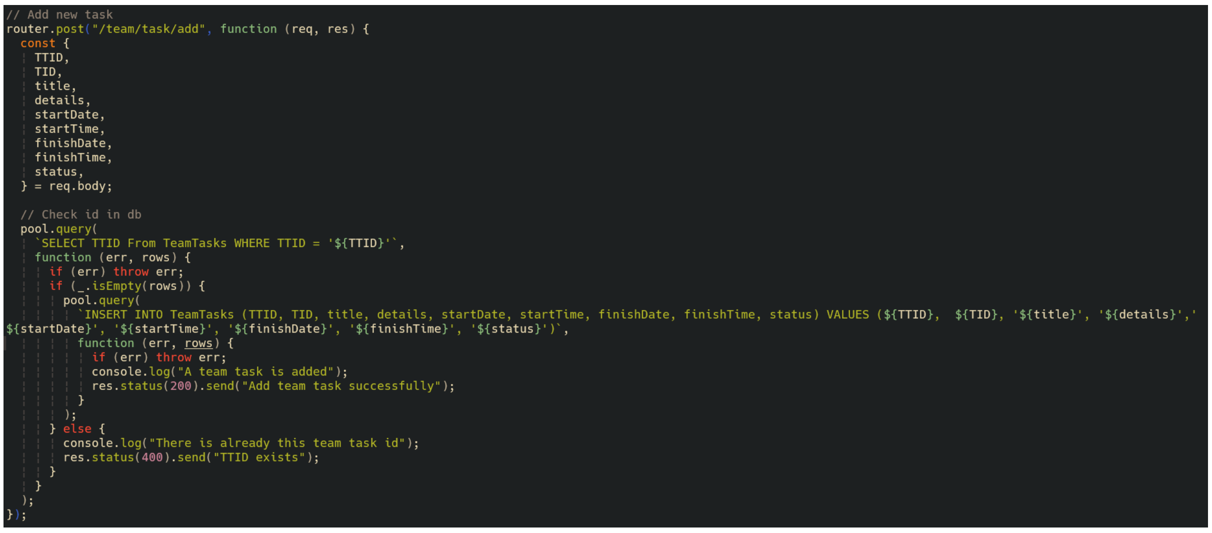
+ “team/employee”: load all members of a team.



```
// Load all employees in a team
router.post("/team/employees", function (req, res) {
  const { TID } = req.body;
  pool.query(
    `SELECT username FROM Employees WHERE TID = '${TID}'`,
    function (err, rows) {
      if (err) throw err;
      return res.status(200).send(rows);
    }
  );
});
```

Figure 5.12: Get all the employees in a team

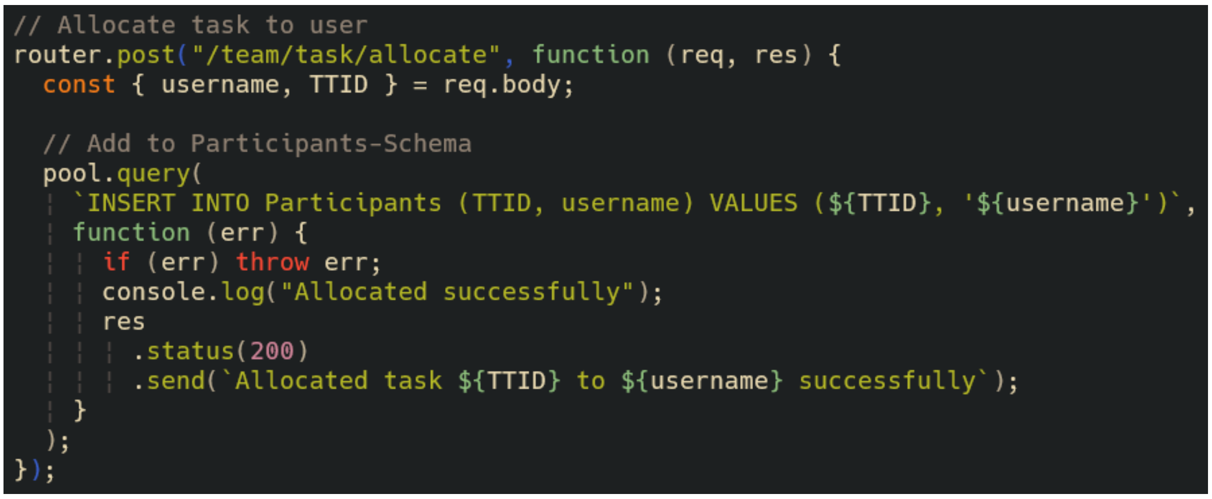
+ “team/task/add”: add new tasks for a team.



```
// Add new task
router.post("/team/task/add", function (req, res) {
  const {
    TTID,
    TID,
    title,
    details,
    startDate,
    startTime,
    finishDate,
    finishTime,
    status,
  } = req.body;

  // Check id in db
  pool.query(
    `SELECT TTID From TeamTasks WHERE TTID = '${TTID}'`,
    function (err, rows) {
      if (err) throw err;
      if (_.isEmpty(rows)) {
        pool.query(
          `INSERT INTO TeamTasks (TTID, TID, title, details, startDate, startTime, finishDate, finishTime, status) VALUES (${TTID},  ${TID}, '${title}', '${details}','
${startDate}', '${startTime}', '${finishDate}', '${finishTime}', '${status}')`,
          function (err, rows) {
            if (err) throw err;
            console.log("A team task is added");
            res.status(200).send("Add team task successfully");
          }
        );
      } else {
        console.log("There is already this team task id");
        res.status(400).send("TTID exists");
      }
    }
  );
});
```

Figure 5.13: Create a new task for teams

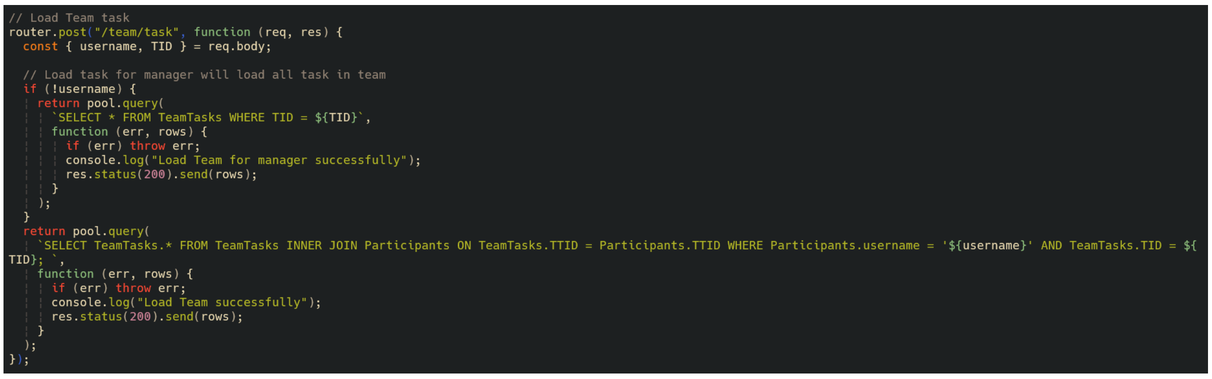
+ "team/tasks/allocate": allocate the new task to a member of the same team.



```
// Allocate task to user
router.post("/team/task/allocate", function (req, res) {
  const { username, TTID } = req.body;

  // Add to Participants-Schema
  pool.query(
    `INSERT INTO Participants (TTID, username) VALUES (${TTID}, '${username}')`,
    function (err) {
      if (err) throw err;
      console.log("Allocated successfully");
      res
        .status(200)
        .send(`Allocated task ${TTID} to ${username} successfully`);
    }
  );
});
```

Figure 5.14: Allocate new tasks for employees

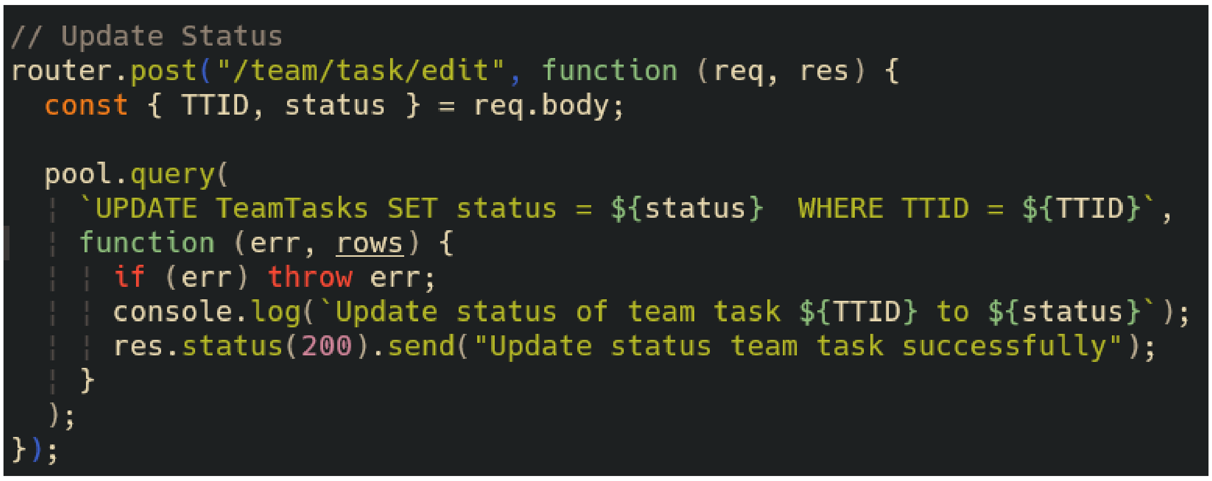
+ "team/task": load tasks for each person in a team.



```
// Load Team task
router.post("/team/task", function (req, res) {
  const { username, TID } = req.body;

  // Load task for manager will load all task in team
  if (!username) {
    return pool.query(
      `SELECT * FROM TeamTasks WHERE TID = ${TID}`,
      function (err, rows) {
        if (err) throw err;
        console.log("Load Team for manager successfully");
        res.status(200).send(rows);
      }
    );
  }
  return pool.query(
    `SELECT TeamTasks.* FROM TeamTasks INNER JOIN Participants ON TeamTasks.TTID = Participants.TTID WHERE Participants.username = '${username}' AND TeamTasks.TID = ${
TID}; `,
    function (err, rows) {
      if (err) throw err;
      console.log("Load Team successfully");
      res.status(200).send(rows);
    }
  );
});
```

Figure 5.15: Get the list of tasks for members in a team

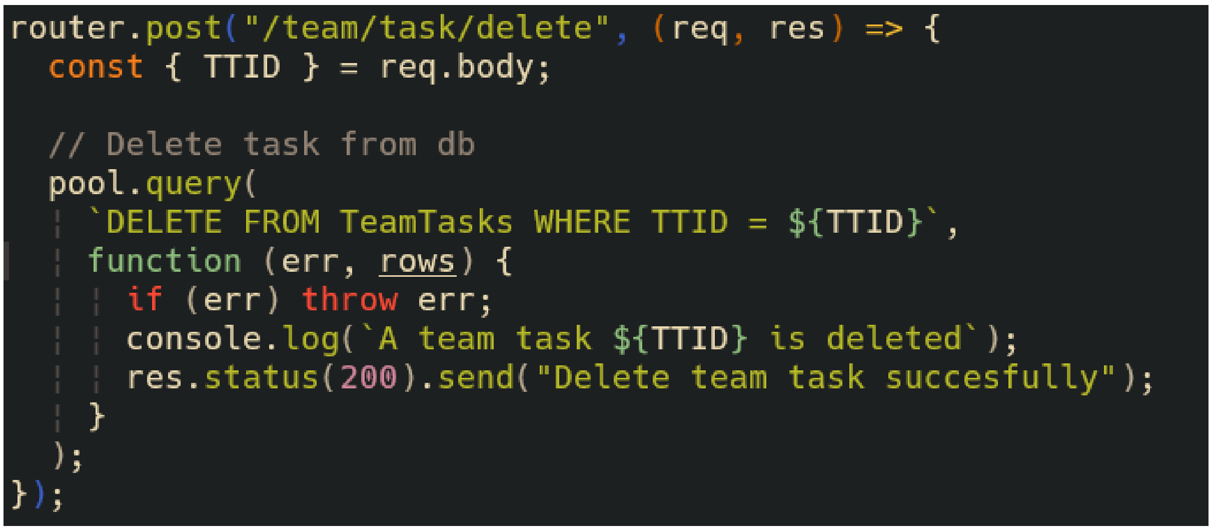
+ "team/task/edit": update the status of a task. This task is done or not.



```
// Update Status
router.post("/team/task/edit", function (req, res) {
  const { TTID, status } = req.body;

  pool.query(
    `UPDATE TeamTasks SET status = ${status}  WHERE TTID = ${TTID}`,
    function (err, rows) {
      if (err) throw err;
      console.log(`Update status of team task ${TTID} to ${status}`);
      res.status(200).send("Update status team task successfully");
    }
  );
});
```

Figure 5.16: Employees update the tasks' progress

+ "team/task/delete": A manager deletes tasks in his/her team.



```
router.post("/team/task/delete", (req, res) => {
  const { TTID } = req.body;

  // Delete task from db
  pool.query(
    `DELETE FROM TeamTasks WHERE TTID = ${TTID}`,
    function (err, rows) {
      if (err) throw err;
      console.log(`A team task ${TTID} is deleted`);
      res.status(200).send("Delete team task succesfully");
    }
  );
});
```

Figure 5.17: Delete tasks in the database

2. Frontend

Firstly, we need to install the development environment for React Native run. You can find all the instructions are clearly described from the React Native document https://reactnative.dev/docs/environment-setup. In the front end, we decide to divide our application into 5 main screens from **section 5.2**. To do this thing we use the library React Navigation to navigate between 5 screens. To post requests and receive responses the data as JSON from the server, we use the library Axios.

- These are some pictures of our application:

+ Log in and log out:

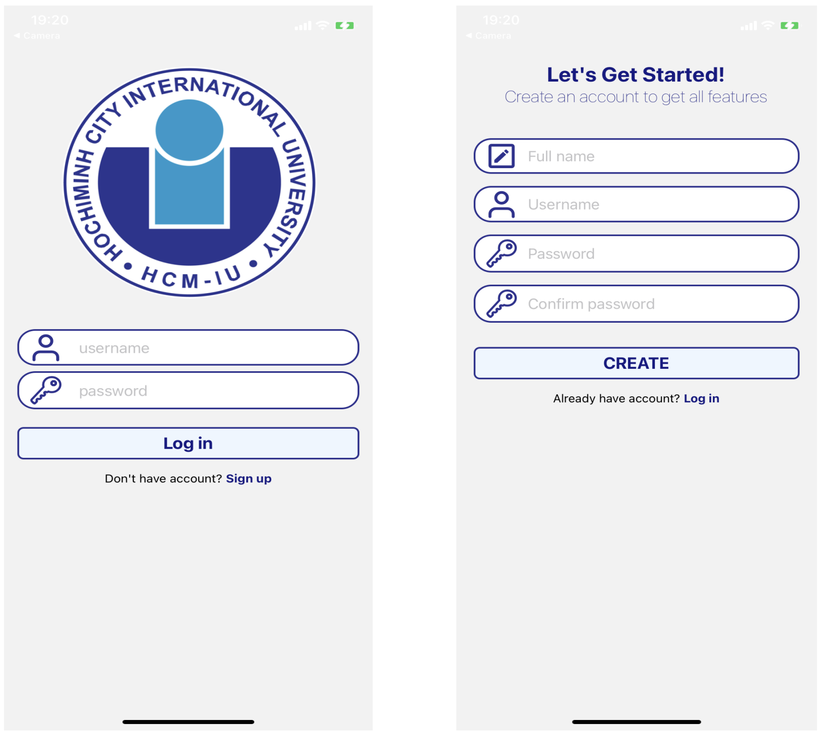
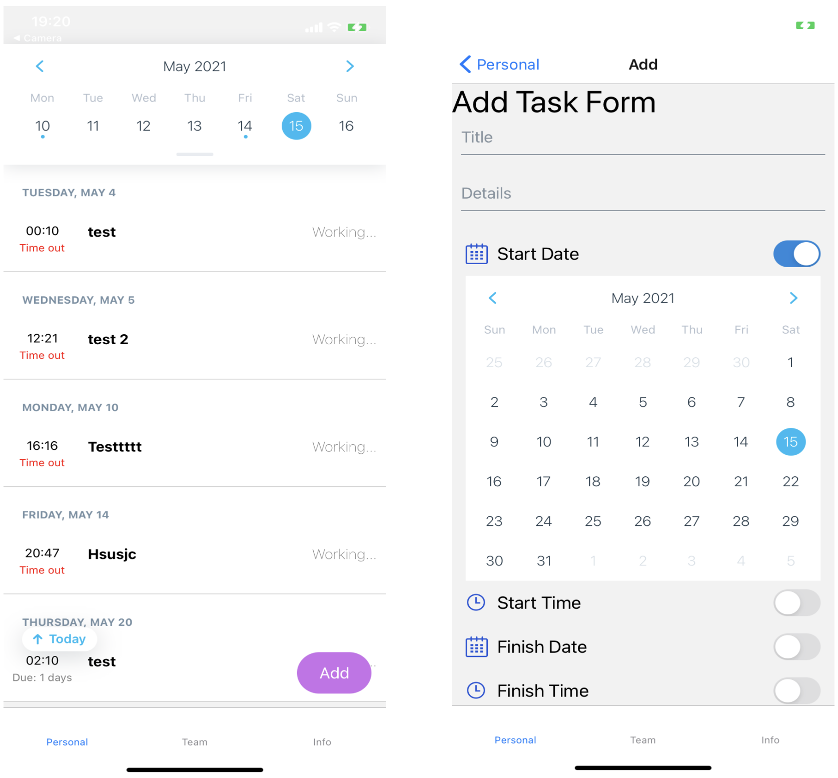

HOCHIMINH CITY INTERNATIONAL UNIVERSITY
HCM-IU
username
password
Log in
Don't have account? Sign up
Let's Get Started!
Create an account to get all features
Full name
Username
Password
Confirm password
CREATE
Already have account? Log in

Figure 5.18: Login and Signup Screen

+ Personal Screen:



TUESDAY, MAY 4
00:10 Time out test Working...

WEDNESDAY, MAY 5
12:21 Time out test 2 Working...

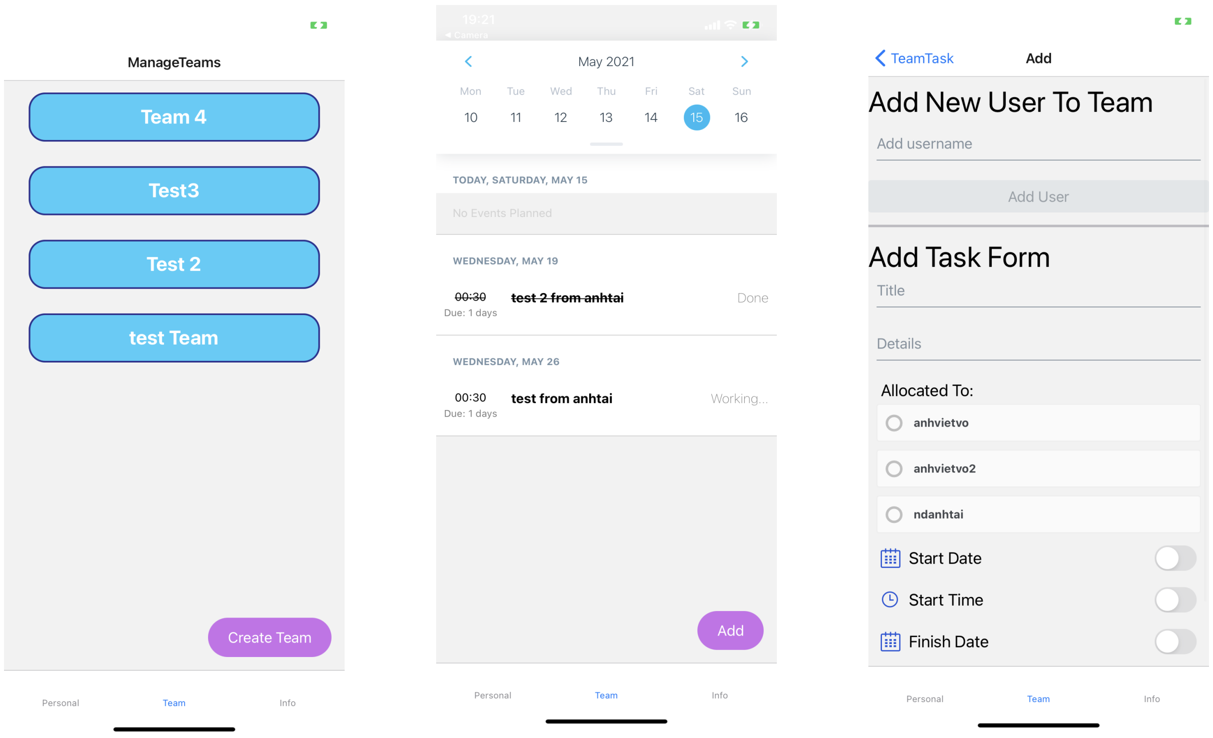
MONDAY, MAY 10
16:16 Time out Testtttt Working...

FRIDAY, MAY 14
20:47 Time out Hsusjc Working...

THURSDAY, MAY 20
Today
02:10 Due: 1 days test

Figure 5.19: Personal Screen

+ Team Screen:



TODAY, SATURDAY, MAY 15

No Events Planned

WEDNESDAY, MAY 19

~~00:30~~ ~~test 2 from anhtai~~ Done
Due: 1 days

WEDNESDAY, MAY 26

00:30 test from anhtai Working...
Due: 1 days

Add New User To Team

Add username

Add Task Form

Title

Details

Allocated To:

- anhvietvo
- anhvietvo2
- ndanhtai

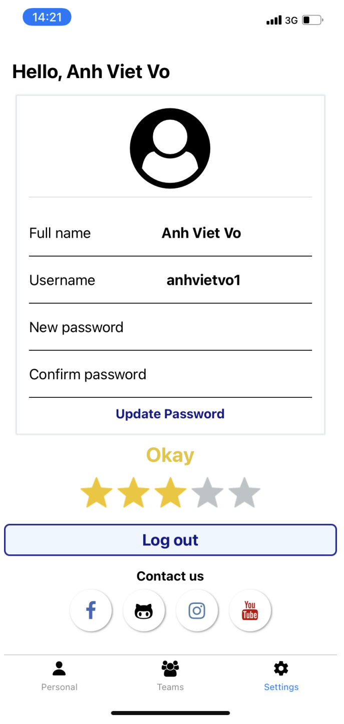
Start Date

Start Time

Finish Date

Figure 5.20: Team Screen

+ Info Screen:



Hello, Anh Viet Vo

Full name	Anh Viet Vo
Username	anhvietvo1
New password	
Confirm password	

Figure 5.21: Info Screen

CHAPTER 6: CONCLUSION

1. Final Thoughts

Overall, after accomplishing this project, we reached higher skills to manipulate databases and hope to apply them fluently into practical circumstances in our careers. However, since we have to complete all the tasks within a short time, many functions, as well as features, have not worked perfectly and needed improvements. Specifically, all things from the backend only work in localhost, so we will try our best to push it to work on the internet as a real application at the nearest time. Also, the frontend at this time is very meager and messy, hence, we will make it more clean and beautiful to attract users. Simultaneously, the backend will be upgraded to adapt to more requests at the same time. This course brings to us much useful knowledge which can help us survive in the future job. We have learned how a real application works in a real-life and the way to make them. We realized that databases are an essential part of the information world. Thanks to databases, technology can develop at a fast tempo nowadays and bring much data to users. The database can optimize the performance of any individual and organization. The importance of databases cannot be denied. We must always try our best to handle data efficiently as much as possible.

2. Future Development

- Backend:
    + At that time, all things only work on localhost, so we will try our best to push it to work on the internet as a real application.
    + We will consider updating Docker for this backend.
    + Improving operating performance to adapt with the increasing demand in the future.
    + We will try to handle more errors that can occur to shutdown our server.
    + Adding more functions such as changing the username, fullname, task name, manager in a team,...
- Frontend:
    + We will design the application again to remove many redundant things.
    + Improving the UI, as well as the UX, for the application to be more friendly with users.
    + Optimize some frontend code to render more smoothly.
    + Changing the way to navigate between screens for users to interact easier.
    + Adding more screens such as the settings screen, feedback screen,...
- Finally, we will try to bring them to the AppStore or CHPlay for users to install from anywhere.

REFERENCES

1. https://reactnative.dev/
2. https://reactnavigation.org/
3. https://github.com/axios/axios
4. https://expressjs.com/
5. https://github.com/sidorares/node-mysql2
6. https://github.com/mmazzarolo/react-native-modal-datetime-picker
7. https://github.com/wix/react-native-calendars
8. https://reactnativeelements.com/docs/
9. https://www.w3schools.com/sql/default.asp
10. https://www.mysql.com/
11. https://www.tutorialspoint.com/mysql/index.htm
12. https://www.javatpoint.com/example-to-connect-to-the-mysql-database
13. https://ngrok.com/
14. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
15. https://www.udemy.com/
16. Many thanks to https://stackoverflow.com/