

Case Study Documentation

1. Abstract:

ITI Simulator is a user-friendly desktop application designed for information technology institutions. It features a centralized database system to manage and implement grading systems efficiently. This application streamlines the process of recording, tracking, and analyzing student grades, providing instructors and students with valuable insights for academic management and decision-making.

1.1. Key Features:

For Students:

- Personal Data Viewing: Students can easily access and view their personal information.
- Program Information: Access to detailed information about their enrolled program.
- Course List: Display of courses included in their program.
- Class Information: View information about their classes, including schedules.
- Performance Report: Access to a comprehensive report showing their academic performance.

For Instructors:

- Personal Data Viewing: Instructors can easily access and view their personal information.
- Grading System Management: Ability to manage the grading system for students enrolled in their classes.
- Qualifications and Experience Management: Manage and update qualifications and experience relevant to their courses.
- Student Performance Reports: Access reports detailing the performance of students in their classes, facilitating effective teaching and monitoring.

2. Requirement Analysis and Design:

User Story :

The institution offers many tracks, and the system should be able to store the track name, track description, business field (software engineering, information systems, content creation, infrastructure and security, and industrial systems), and the establishment date for each track.

Each track contains many programs. The system should be able to store the program name, description, category (9 months or 3 months), job profiles related to this program, and establishment date for each program.

Programs initiate many intakes based on their category. The 9-month program starts from October to the end of June the following year. The system should store the start and end dates of each intake, with each intake related to only one program.

Each track has many instructors working for it, with one of them being the supervisor of the track. The system should store data about the instructors, such as name, SSN, address, phone number, date of birth, email, and qualifications.

Some instructors may have graduated from our intakes.

Each program contains a list of courses associated with it, and each course may be associated with many programs. The system should store data about courses, including course name, credit hours, description, and syllabus.

Each instructor may be experienced in many courses and can teach them for intakes.

Each course has prerequisite courses.

The system stores data about students, including name, address, email, phone number, faculty of graduation, and date of birth.

Each intake has many students joining it, with each student belonging to just one intake.

Intakes initiate classes that teach courses, with the instructor assigned to teach that course. The system should store data about each class, such as start date, end date, type (online or offline), project percentage, test percentage, and assignment percentage. One course can be taught in many classes for different intakes, and an instructor can teach many courses in different classes.

Each class establishes a schedule. The system should store data about each class schedule, including the day of the week, start time, and end time.

Students attend one day schedule, with attendance taken by the instructor. The attendance status represents whether the student is absent or present.

Each class has many assignments, each with a description and type (project or task), and a deadline. The system should store that information.

Student grades in each assignment are stored in the system, graded by the instructor after the student submits, with grades ranging between 0.0 to 4.0.

Each class has only one test, and students must pass that test. Each test has a grade like an assignment.

The system stores grades for mapping the final grade for the student, such as grade A, A-, etc.

The final grade is calculated by multiplying the test percentage with its grade and summing all student grades in the assignment with its

percentage, and multiplying the project percentage with its grade and summing all together to store the final grade point for that class.

Then, the system maps that point using the grade table to automatically update the student's final grade by the end of each class. This is akin to a GPA, where the points are multiplied by hours and divided by the total hours for all classes.

The system has a simple login window. Each user is either an instructor or student and must check for validation of their data and authentication before accessing the main screen. In case of forgetting the password, users must connect to the supervisor of the track.

2.1. Functional Requirement:

- **Track Management:** The system shall allow administrators to create, read, update, and delete tracks. Each track shall contain fields for track name, description, business field, and establishment date.
- **Program Management:** The system shall enable administrators to manage programs within each track. Each program shall have fields for program name, description, category (9 months or 3 months), job profiles, and establishment date.
- **Intake Management:** The system shall facilitate the creation of intakes for each program, with start and end dates. Intakes shall be associated with one program and categorized based on their duration.
- **Instructor Management:** Administrators shall be able to manage instructors assigned to each track. Instructor profiles shall include name, SSN, contact information, date of birth, email, and qualifications.
- **Course Management:** The system shall allow administrators to manage courses associated with each program. Course

details shall include name, credit hours, description, and syllabus.

- Class Management: The system shall enable administrators to schedule classes for each intake. Class details shall include start date, end date, type (online or offline), and grading percentages.
- Attendance Tracking: Instructors shall be able to record student attendance for each class session. Attendance status shall be marked as present or absent.
- Assignment Management: The system shall support the creation and management of assignments for each class. Assignments shall include descriptions, types (project or task), and deadlines.
- Grading System: Instructors shall be able to grade student assignments and tests, ranging from 0.0 to 4.0. Final grades for classes shall be calculated based on assignment and test grades, with mappings to letter grades (e.g., A, A-, B+).
- User Authentication: The system shall provide a login window for users, distinguishing between instructors and students. Users shall undergo validation and authentication processes before accessing the main screen.

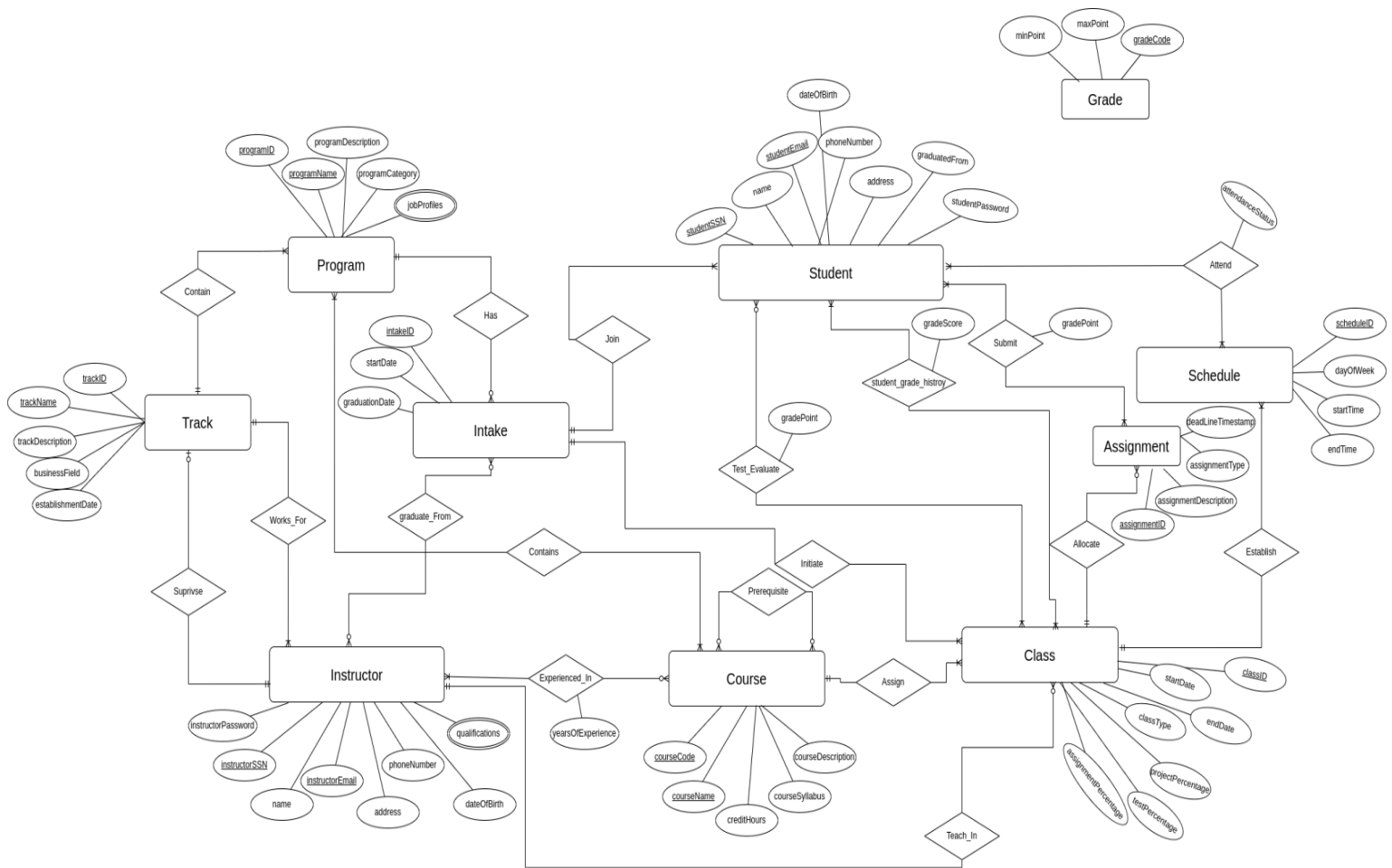
2.2. Non-Functional Requirement:

- **Performance:** The system shall respond to user interactions within 2 seconds. Database queries shall execute efficiently to minimize loading times.
- **Reliability:** The system shall have a backup mechanism in place to prevent data loss in case of system failures. Scheduled maintenance tasks, such as disk monitoring and database backups, shall be automated to ensure system reliability.
- **Usability:** The user interface shall be intuitive and user-friendly, requiring minimal training for users to navigate and perform tasks. Error messages shall be descriptive and actionable to assist users in troubleshooting issues effectively.
- **Compatibility:** The system shall be compatible with modern web browsers and operating systems, ensuring accessibility for a diverse user base.

3. System Design:

3.1. Database Design:

First step in the database design based on the requirements is to design the ERD model



1. Track Entity:

- ❖ **Attributes:** Track ID (Unique), Track Name(Unique), Track Description, Business Field('content developments', 'industrial systems', 'information system', 'infrastructure and security', 'software engineering'), Establishment Date
- ❖ **Relationships:** One track can have many programs, and just one instructor supervises one track.

2. Program Entity:

- ❖ Attributes: Program ID (Unique), Program Name(Unique), Description, Category (9 months or 3 months), Establishment Date
- ❖ Relationships: One program belongs to one track. One program can have many intakes. One program can have many courses. One program has many job profiles and job profiles may be in many programs.

3. Intake Entity:

- ❖ Attributes: Intake ID (Unique), Start Date(Unique), End Date(Unique).
- ❖ Relationships: One intake belongs to one program. One intake has many students.

4. Instructor Entity:

- ❖ Attributes: Instructor ID (Unique), Name (Unique), SSN (Unique), Address, Phone Number (Unique), Date of Birth, Email, Qualifications
- ❖ Relationships: One instructor can work for one track. One instructor can teach many courses. Some instructors may have graduated from intakes, instructor has many qualifications and certifications, the instructor can may have experience in teaching many courses.

5. Course Entity:

- ❖ Attributes: Course ID (Unique), Course Name(Unique), Credit Hours, Description, Syllabus
- ❖ Relationships: One course can belong to many programs. One course can have many prerequisite courses.

6. Student Entity:

- ❖ Attributes: Student ID (Unique), Name(Unique), Address, Email(Unique), Phone Number(Unique), Faculty of Graduation, Date of Birth
- ❖ Relationships: One student belongs to one intake. One student attends many classes and submits many assignments for one class.

7. Class Entity:

- ❖ Attributes: Class ID (Unique), Start Date, End Date, Type (Online or Offline), Project Percentage, Test Percentage, Assignment Percentage
- ❖ Relationships: One class initiated from one intake. One class can have many assignments. One class has many students attending, one class has one test and students get grade points on it.

8. Class Schedule Entity:

- ❖ Attributes: Schedule ID (Unique), Day of the Week, Start Time, End Time
- ❖ Relationships: One class schedule belongs to one class, and one class has many scheduled times.

9. Assignment Entity:

- ❖ Attributes: Assignment ID (Unique), Description, Type (Project or Task), Deadline
- ❖ Relationships: One assignment belongs to one class, and one assignment is submitted by many students.

10. Grade Entity:

- ❖ Attributes: Grade ID (Primary Key), max point, min point, (0.0 to 4.0)
- ❖ Relationships: is just lookup/helper table.

The second step is to build the logical schema and mapping the erd model to relational schema design

Relational Mapping:

Track (TRACK_ID, TRACK_NAME, TRACK_DESCRIPTION, BUSINESS_FIELD, ESTABLISHMENT_DATE, SUPERVISOR_SSN)

Student (STUDENT_SSN, STUDENT_NAME, STUDENT_EMAIL, DATE_OF_BIRTH, PHONE_NUMBER, ADDRESS, FACULTY, INTAKE_ID, STUDENT_PASSWORD, CUMULATIVE_GRADE)

Schedule (SCHEDULE_ID, DAY_OF_WEEK, START_TIME, END_TIME, CLASS_ID)

Qualification (QUALIFICATION_ID, QUALIFICATION_NAME, QUALIFICATION_DESCRIPTION)

Program (PROGRAM_ID, PROGRAM_NAME, PROGRAM_DESCRIPTION, PROGRAM_CATEGORY, PROGRAM_ESTABLISHMENT_DATE, TRACK_ID)

Job Profile (JOB_ID, JOB_PROFILE_NAME, JOB_DESCRIPTION) I

intake (INTAKE_ID, START_DATE, END_DATE, PROGRAM_ID)

Instructor (INSTRUCTOR_SSN, INSTRUCTOR_NAME, INSTRUCTOR_EMAIL, ADDRESS, PHONE, DATE_OF_BIRTH, TRACK_ID, INSTRUCTOR_PASSWORD, INTAKE_ID)

Grade (GRADE_CODE, MAX_POINT, MIN_POINT)

Course (COURSE_CODE, COURSE_NAME, CREDIT_HOURS, COURSE_SYLLABUS, COURSE_DESCRIPTION)

Class Test (CLASS_ID, STUDENT_SSN, GRADE_POINT)

Class Final Grade (CLASS_ID, STUDENT_SSN, GRADE)

Class (CLASS_ID, CLASS_TYPE, START_DATE, END_DATE, PROJECT_PERCENTAGE, TEST_PERCENTAGE, ASSIGNMENT_PERCENTAGE, INSTRUCTOR_SSN, COURSE_CODE, INTAKE_ID)

Attendance (STUDENT_SSN, SCHEDULE_ID, ATTENDANCE_STATUS)

Assignment Grade (ASSIGNMENT_ID, STUDENT_SSN, GRADE_POINT)

Assignment (ASSIGNMENT_ID, ASSIGNMENT_DESCRIPTION, ASSIGNMENT_TYPE, DEADLINE, CLASS_ID)

Prerequisites (COURSE_CODE, PREREQUISITE_CODE)

Course Program (COURSE_CODE, PROGRAM_ID)

Course Instructor (COURSE_CODE, INSTRUCTOR_SSN)

Job Profile Program(JOB_ID, PROGRAM_ID)

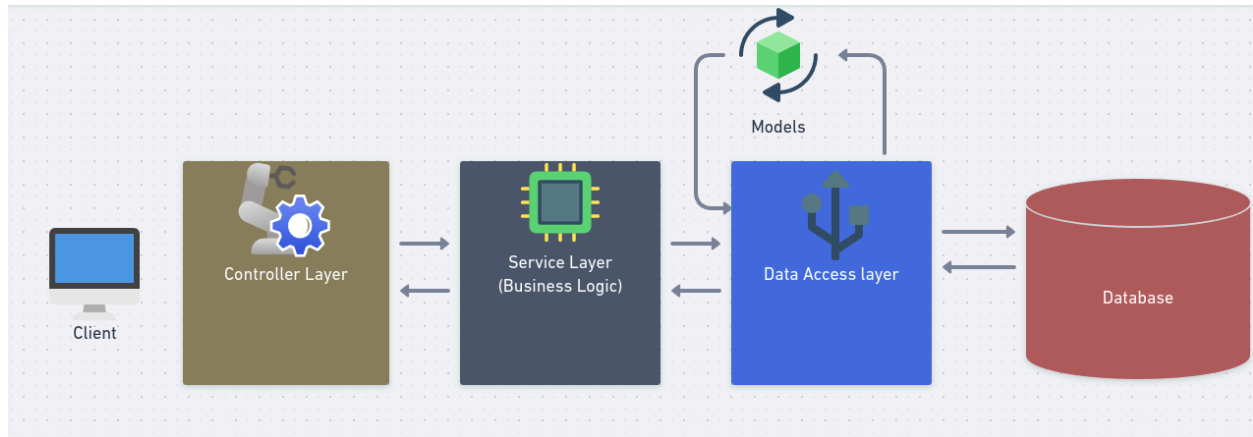
Normalization:

In this step i test my design to be normalized form.

- ❖ All tables have a primary key identifying them.
- ❖ Tables are organized to minimize data redundancy and adhere to normalization rules.
- ❖ Relationships between tables are maintained through foreign key constraints, ensuring data integrity.
- ❖ No repeating groups or multi-valued dependencies are present.
- ❖ Each attribute within a table is dependent on the primary key.

But my design pitfalls in that there are many relationships between student and class that the grade point is redundant in many tables like in test and the student grade, and also may be nulls in the intake ID column in the table of the instructor because maybe not graduate from our institution.

3.2. Application Design:



I have adopted a structured approach inspired by the Model-View-Controller (MVC) pattern and key software engineering principles such as separation of concerns and single responsibility. This approach ensures the clarity, maintainability, and scalability of the application.

my design consists of distinct layers, each with its own specific responsibilities:

1. User Interface (UI) Layer:

- This layer encompasses the graphical user interface elements, including buttons and other UI components, designed for user interaction.
- Its primary role is to provide a user-friendly interface for interacting with the application.

2. Controller Layer:

- The controller layer is responsible for managing the flow of events triggered by user interactions in the UI layer.
- It determines how events are handled and orchestrates the overall flow of the application.

3. Service (or Logic) Layer:

- This layer contains the business logic of the application, including data validation and formatting.
- It ensures that business rules are enforced and implements the necessary logic to process data effectively.

4. Data Access Layer:

- The data access layer is responsible for handling database connections and executing queries to retrieve or manipulate data.

- It abstracts the underlying database operations, providing a unified interface for interacting with the database.

5. Model Layer:

- The model layer governs how objects within the application communicate and interact.
- It facilitates message passing between different components and ensures seamless data exchange throughout the system.

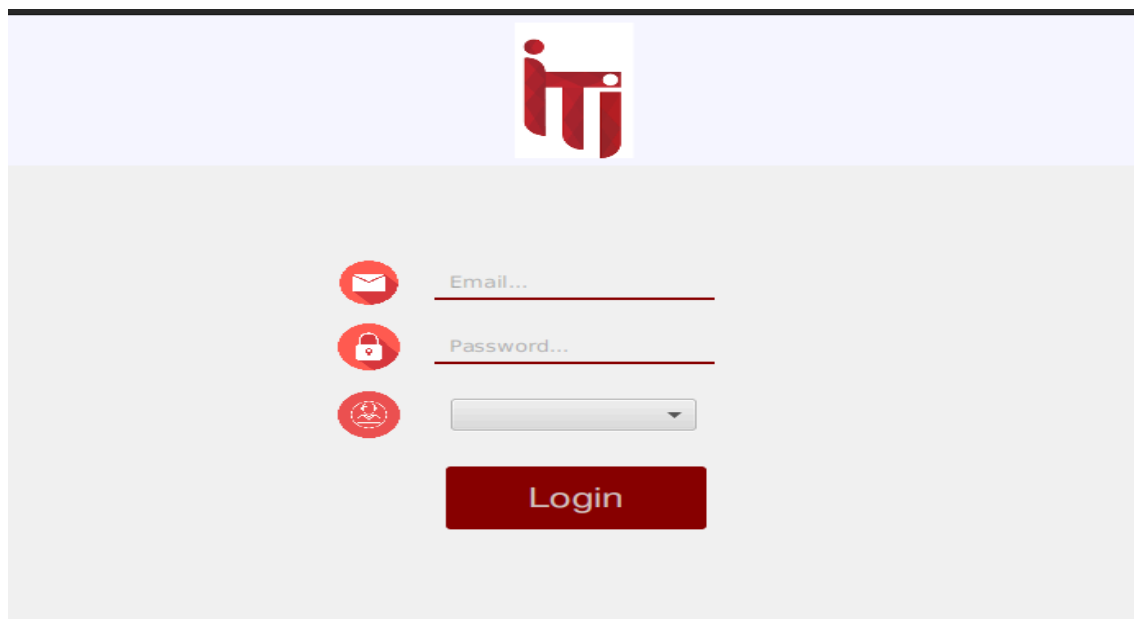
By adhering to this design structure, I hope I achieve a modular and well-organized architecture that promotes code reusability, maintainability, and scalability. Each layer has a clear and distinct purpose, allowing for easy separation of concerns and enabling efficient collaboration among development teams. This design approach also fosters flexibility, making it easier to adapt to evolving requirements and technological advancements in the future.

3.3. UI Design:

Login view:

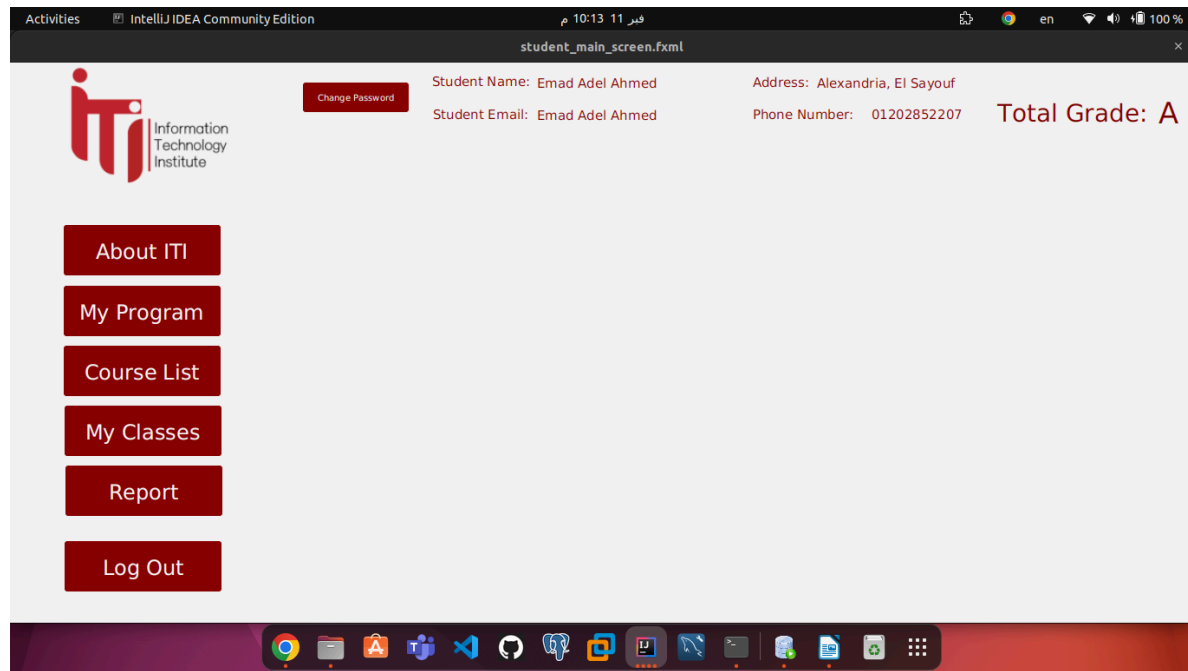
1. Login View:

- The login view serves as the entry point for all users and remains consistent across different roles.
- Designed with a minimalist approach, it features a clean layout with red and white color palette for a visually appealing interface.
- The login form is prominently displayed, allowing users to input their credentials easily.
- Clear and concise instructions guide users through the login process, ensuring a seamless experience.



The image shows a login view UI design. At the top, there is a light blue header bar containing a red logo that resembles a stylized 'iti'. Below the header, the main content area has a light gray background. On the left side of this area, there are three red circular icons: an envelope (email), a padlock (password), and a person (user profile). To the right of these icons are three input fields: the first is labeled 'Email...' and has a red underline; the second is labeled 'Password...' and also has a red underline; the third is a dropdown menu with a gray background and a small downward arrow. Below these input fields is a large red button with the word 'Login' in white text.

Main Screen View:



2. Main Screen View:

- The main screen view dynamically adjusts based on the user's role, presenting tailored content and functionality.
- Utilizing a similar red and white color scheme, the main screen maintains visual continuity with the login view.
- A menu bar positioned on the left-hand side offers easy navigation to different sections of the application, providing quick access to relevant features.
- At the top of the screen, user-specific data is displayed, such as profile information and notifications, enhancing personalization and user engagement.
- Helper buttons strategically placed throughout the interface offer additional functionality, enhancing user productivity and workflow efficiency.

4. Development:

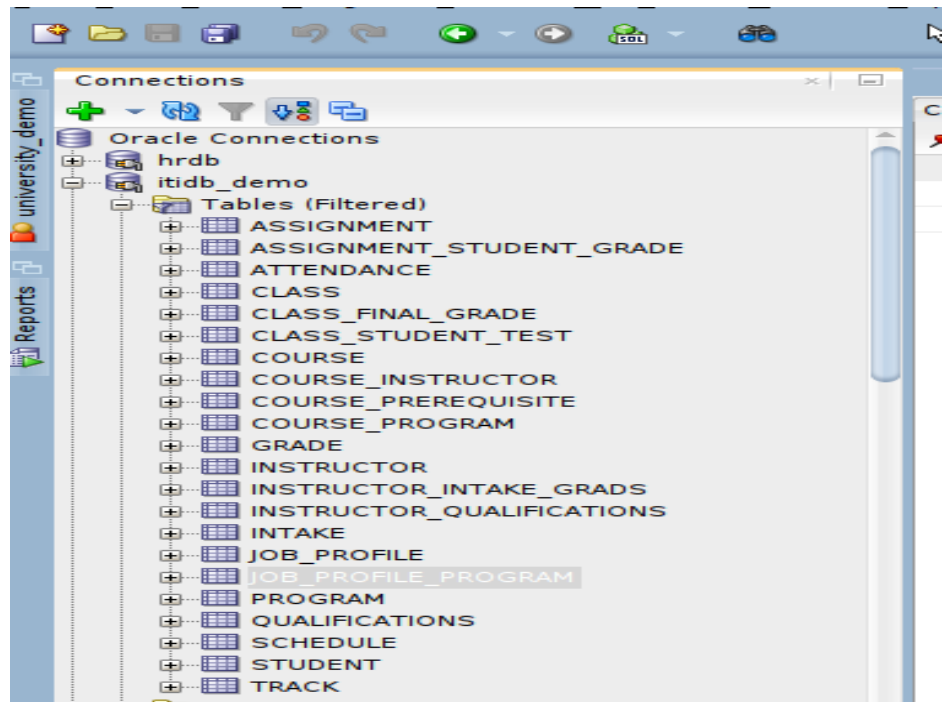
4.1. Technology Stack:

- Oracle Database (Version 11.2 xg): Utilized for maintaining a centralized database, Oracle Database provides a reliable platform for storing and managing data. It serves as the backbone for handling CRUD operations efficiently.
- Java Development Language(JDK11 and Maven build tool): Java serves as the primary language for application development, offering a robust environment for implementing business logic and handling data flow within the system. Through Java, we ensure scalability, performance, and platform independence.
- JavaFX with Scene Builder: JavaFX, coupled with Scene Builder, facilitates the creation of a dynamic and visually appealing Graphical User Interface (GUI) for the application. Scene Builder streamlines UI design, enhancing productivity and enabling seamless integration with Java code.
- Bash Scripting: Bash scripting is employed for automation tasks, including disk monitoring and scheduling database backups. Its versatility and ease of use make it an ideal choice for managing routine operations efficiently.
- IntelliJ IDEA IDE: IntelliJ IDEA serves as the Integrated Development Environment (IDE) for this project. Its robust features and seamless integration with Java and other technologies enhance developer productivity and streamline the development process.

4.2. Database Implementation:

Physical Schema

Tables



1. Track

- TRACK_ID (NUMBER(4,0))
- TRACK_NAME (VARCHAR2(100 BYTE))
- TRACK_DESCRIPTION (VARCHAR2(4000 BYTE))
- BUSINESS_FIELD (VARCHAR2(100 BYTE))
- ESTABLISHMENT_DATE (DATE)
- SUPERVISOR_SSN (CHAR(14 BYTE))

2. Student

- STUDENT_SSN (CHAR(14 BYTE))
- STUDENT_NAME (VARCHAR2(100 BYTE))
- STUDENT_EMAIL (VARCHAR2(100 BYTE))
- DATE_OF_BIRTH (DATE)
- PHONE_NUMBER (CHAR(13 BYTE))
- ADDRESS (VARCHAR2(255 BYTE))
- FACULTY (VARCHAR2(100 BYTE))

- INTAKE_ID (VARCHAR2(100 BYTE))
- STUDENT_PASSWORD (VARCHAR2(20 BYTE))
- CUMULATIVE_GRADE (VARCHAR2(3 BYTE))

3. Schedule

- SCHEDULE_ID (NUMBER)
- DAY_OF_WEEK (VARCHAR2(20 BYTE))
- START_TIME (TIMESTAMP(6))
- END_TIME (TIMESTAMP(6))
- CLASS_ID (NUMBER)

4. Qualifications

- QUALIFICATION_ID (NUMBER)
- QUALIFICATION_NAME (VARCHAR2(100 BYTE))
- QUALIFICATION_DESCRIPTION (VARCHAR2(255 BYTE))

5. Program

- PROGRAM_ID (NUMBER(6,0))
- PROGRAM_NAME (VARCHAR2(100 BYTE))
- PROGRAM_DESCRIPTION (VARCHAR2(4000 BYTE))
- PROGRAM_CATEGORY (VARCHAR2(50 BYTE))
- PROGRAM_ESTABLISHMENT_DATE (DATE)
- TRACK_ID (NUMBER(4,0))

6. Job_Profile_Program

- JOB_ID (NUMBER)
- PROGRAM_ID (NUMBER)

7. Job_Profile

- JOB_ID (NUMBER)
- JOB_PROFILE_NAME (VARCHAR2(100 BYTE))
- JOB_DESCRIPTION (VARCHAR2(255 BYTE))

8. Intake

- INTAKE_ID (VARCHAR2(100 BYTE))

- START_DATE (DATE)
- END_DATE (DATE)
- PROGRAM_ID (NUMBER(6,0))

9. Instructor_Qualifications

- INSTRUCTOR_SSN (CHAR(14 BYTE))
- QUALIFICATION_ID (NUMBER)

10. Intake_Instructor_Graduates

- INTAKE_ID (VARCHAR2(100 BYTE))
- INSTRUCTOR_SSN (CHAR(14 BYTE))

11. Instructor

- INSTRUCTOR_SSN (CHAR(14 BYTE))
- INSTRUCTOR_NAME (VARCHAR2(100 BYTE))
- INSTRUCTOR_EMAIL (VARCHAR2(100 BYTE))
- ADDRESS (VARCHAR2(255 BYTE))
- PHONE (NUMBER)
- DATE_OF_BIRTH (DATE)
- TRACK_ID (NUMBER(4,0))
- INSTRUCTOR_PASSWORD (VARCHAR2(20 BYTE))

12. Grade

- GRADE_CODE (VARCHAR2(3 BYTE))
- MAX_POINT (NUMBER)
- MIN_POINT (NUMBER)

13. Program_Course

- COURSE_CODE (NUMBER)
- PROGRAM_ID (NUMBER(6,0))

14. Course_Prerequisites

- COURSE_CODE (NUMBER)
- PREREQUISITE_CODE (NUMBER)

15. Course_Instructor_Experience

- COURSE_CODE (NUMBER)
- INSTRUCTOR_SSN (CHAR(14 BYTE))
- YEARS_OF_EXPERIENCE (NUMBER(2,0))

16. Course

- COURSE_CODE (NUMBER)
- COURSE_NAME (VARCHAR2(100 BYTE))
- CREDIT_HOURS (NUMBER)
- COURSE_SYLLABUS (VARCHAR2(1000 BYTE))
- COURSE_DESCRIPTION (VARCHAR2(1000 BYTE))

17. Class_Test

- CLASS_ID (NUMBER)
- STUDENT_SSN (CHAR(14 BYTE))
- GRADE_POINT (NUMBER)

18. Class_Final_Grade

- CLASS_ID (NUMBER)
- STUDENT_SSN (CHAR(14 BYTE))
- GRADE (NUMBER(5,2))

19. Class

- CLASS_ID (NUMBER)
- CLASS_TYPE (VARCHAR2(50 BYTE))
- START_DATE (DATE)
- END_DATE (DATE)
- PROJECT_PERCENTAGE (NUMBER)
- TEST_PERCENTAGE (NUMBER)
- ASSIGNMENT_PERCENTAGE (NUMBER)
- INSTRUCTOR_SSN (CHAR(14 BYTE))
- COURSE_CODE (NUMBER)
- INTAKE_ID (VARCHAR2(100 BYTE))

20. Attendance

- STUDENT_SSN (CHAR(14 BYTE))

- SCHEDULE_ID (NUMBER)
- ATTENDANCE_STATUS (NUMBER(1,0))

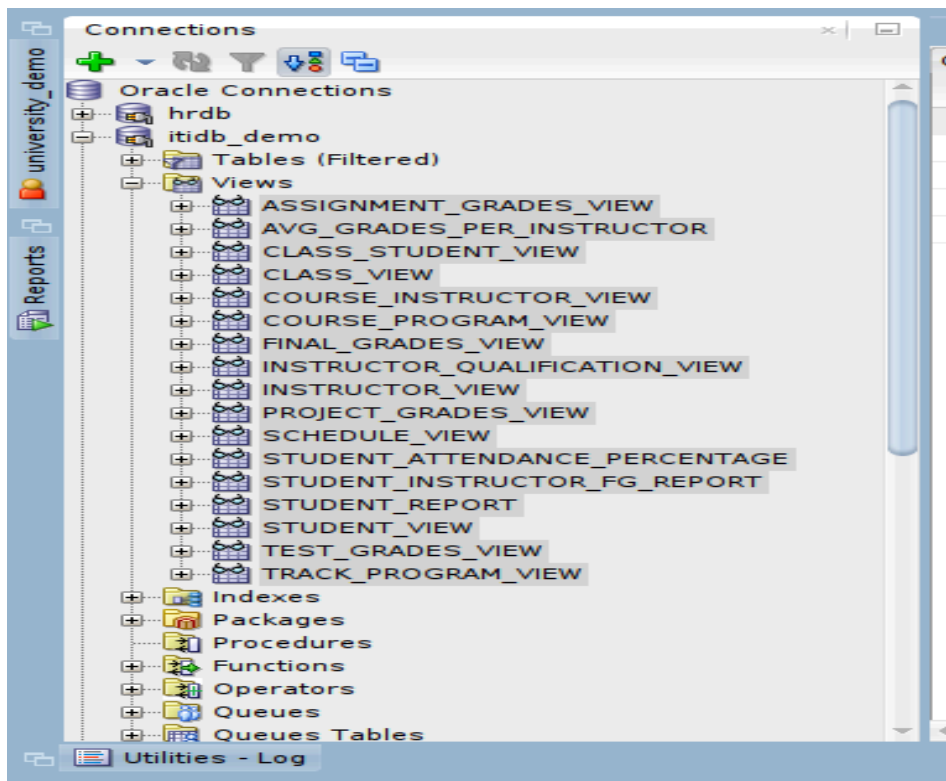
21. Assignment_Grades

- ASSIGNMENT_ID (NUMBER)
- STUDENT_SSN (CHAR(14 BYTE))
- GRADE_POINT (NUMBER)

22. Assignment

- ASSIGNMENT_ID (NUMBER)
- ASSIGNMENT_DESCRIPTION (VARCHAR2(255 BYTE))
- ASSIGNMENT_TYPE (VARCHAR2(50 BYTE))
- DEADLINE (TIMESTAMP(6))
- CLASS_ID (NUMBER)

Views



Views play the role of internal schema between the application layer and the database layer for applying the data abstraction principle.

1. ASSIGNMENT_GRADES_VIEW: This view displays all assignment grades for students across various classes.

2. AVG_GRADES_PER_INSTRUCTOR: This view provides the average grades given by each instructor across all classes.

3. CLASS_STUDENT_VIE: This view shows the list of students enrolled in each class.

4. CLASS_VIEW: This view presents detailed information about each class, including the instructor, course, schedule, and other relevant details.

5. COURSE_INSTRUCTOR_VIEW: This view displays the instructors assigned to teach each course.

6. COURSE_PROGRAM_VIEW: This view shows the mapping between courses and programs, indicating which courses are included in each program.

7. FINAL_GRADES_VIEW: This view displays the final grades of students for each class.

8. INSTRUCTOR_QUALIFICATION_VIEW: This view shows the qualifications of each instructor.

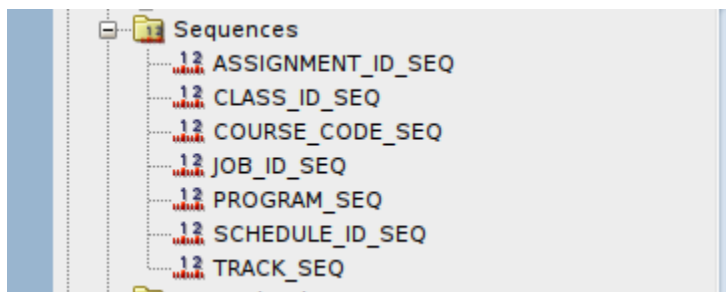
9. INSTRUCTOR_VIEW: This view provides detailed information about each instructor, including their contact details, qualifications, and assigned tracks.

10. PROJECT_GRADES_VIEW: This view displays the grades specifically for project assignments across all classes.

11. SCHEDULE_VIEW: This view shows the schedule of classes, including the day, time, and location of each class.

12. STUDENT_ATTENDANCE_PERCENTAGE: This view calculates and displays the attendance percentage for each student in each class.
13. STUDENT_INSTRUCTOR_FG_REPORT: This view generates a report showing the final grades of students for each instructor.
14. STUDENT_REPORT: This view provides a comprehensive report about each student, including their grades, attendance, and other relevant information.
15. STUDENT_VIEW: This view may provides detailed information about each student, including their personal details, enrollment status, and academic performance.
16. TEST_GRADES_VIEW: This view displays the grades specifically for test assignments across all classes.
17. TRACK_PROGRAM_VIEW: This view shows the mapping between tracks and programs, indicating which programs are offered under each track.

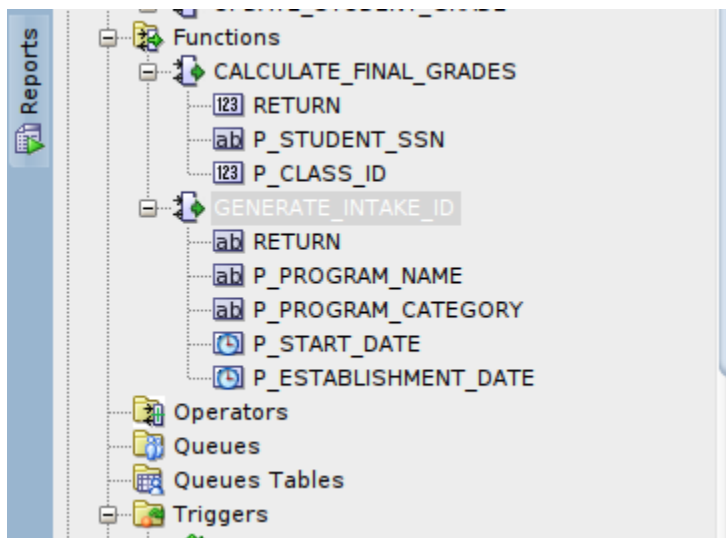
Sequences:



1. ASSIGNMENT_ID_SEQ:: This sequence used to generate unique IDs for assignments.
2. CLASS_ID_SEQ: This sequence is used to generate unique IDs for classes.
3. COURSE_CODE_SEQ: This sequence is used to generate unique codes for courses.
4. JOB_ID_SEQ: This sequence is used to generate unique IDs for job profiles.

5. PROGRAM_SEQ: This sequence is used to generate unique IDs for programs.
6. SCHEDULE_ID_SEQ: This sequence is used to generate unique IDs for schedules or timetable entries.
7. TRACK_SEQ: This sequence is used to generate unique IDs for tracks or academic programs.

Functions:



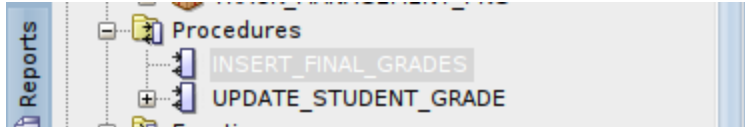
GENERATE_INTAKE_ID

This function generates a program intake ID based on program details: name, category, start date, and establishment date. It calculates the year difference for '9 months program' or appends the month for '3 months program' to the name. Finally, it returns the constructed intake ID as a string.

CALCULATE_FINAL_GRADES

This function calculates the final grade for a student in a class based on their test scores, assignment grades, and project grades. It retrieves the percentage weights for each assessment type from the class table. Then, it computes the weighted grades for tests, assignments, and projects using the corresponding percentages. Finally, it sums up the weighted grades to determine the student's total grade for the class.

Procedures:



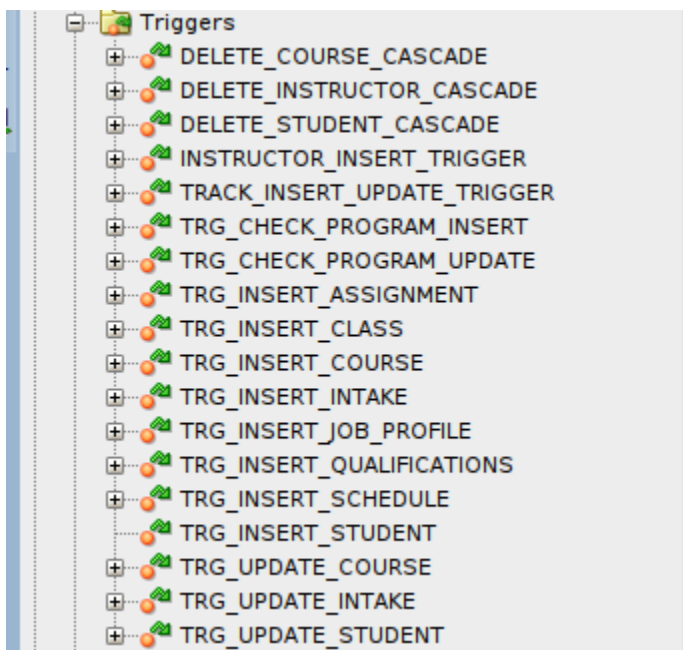
INSERT_FINAL_GRADES

This procedure iterates over classes without final grades, then for each student in those classes, it calculates and inserts the final grade using the `calculate_final_grades` function. After all insertions, it commits the transaction, prints a success message, or rolls back and prints an error if an exception occurs.

UPDATE_STUDENT_GRADE

This procedure calculates the final grade for each student based on their class final grades, then maps it to a grade code and updates the cumulative grade in the student table. It commits the transaction upon completion and prints a success message, handling exceptions by printing an error message and rolling back the transaction if any occur.

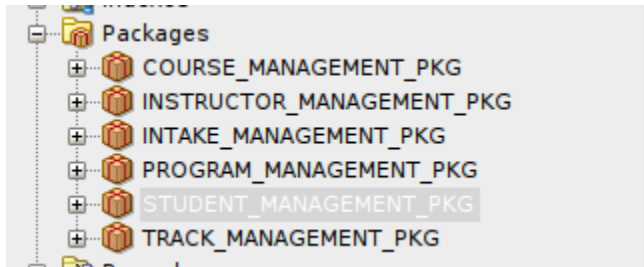
Triggers:



1. DELETE_COURSE_CASCADE: Deletes related records when a course is deleted.
2. DELETE_INSTRUCTOR_CASCADE: Deletes related records when an instructor is deleted.
3. DELETE_STUDENT_CASCADE: Deletes related records when a student is deleted.
4. INSTRUCTOR_INSERT_TRIGGER: Performs actions upon insertion of an instructor.

5. TRACK_INSERT_UPDATE_TRIGGER: Handles insert and update operations on the track table.
6. TRG_CHECK_PROGRAM_INSERT: Validates insert operations on the program table.
7. TRG_CHECK_PROGRAM_UPDATE: Validates update operations on the program table.
8. TRG_INSERT_ASSIGNMENT: Executes actions upon insertion of an assignment record.
9. TRG_INSERT_CLASS: Handles insert operations on the class table.
10. TRG_INSERT_COURSE: Performs actions upon insertion of a course.
11. TRG_INSERT_INTAKE: Executes actions upon insertion of an intake record.
12. TRG_INSERT_JOB_PROFILE: Handles insert operations on the job profile table.
13. TRG_INSERT_QUALIFICATIONS: Performs actions upon insertion of qualifications.
14. TRG_INSERT_SCHEDULE: Executes actions upon insertion of a schedule record.
15. TRG_INSERT_STUDENT: Handles insert operations on the student table.
16. TRG_UPDATE_COURSE: Performs actions upon update of a course record.
17. TRG_UPDATE_INTAKE: Handles update operations on the intake table.
18. TRG_UPDATE_STUDENT: Executes actions upon update of a student record.

Packages:



COURSE_MANAGEMENT_PKG: Manages course records with procedures for insert, delete, and update.

INSTRUCTOR_MANAGEMENT_PKG: Manages instructor data, offering procedures for insert, delete, and update.

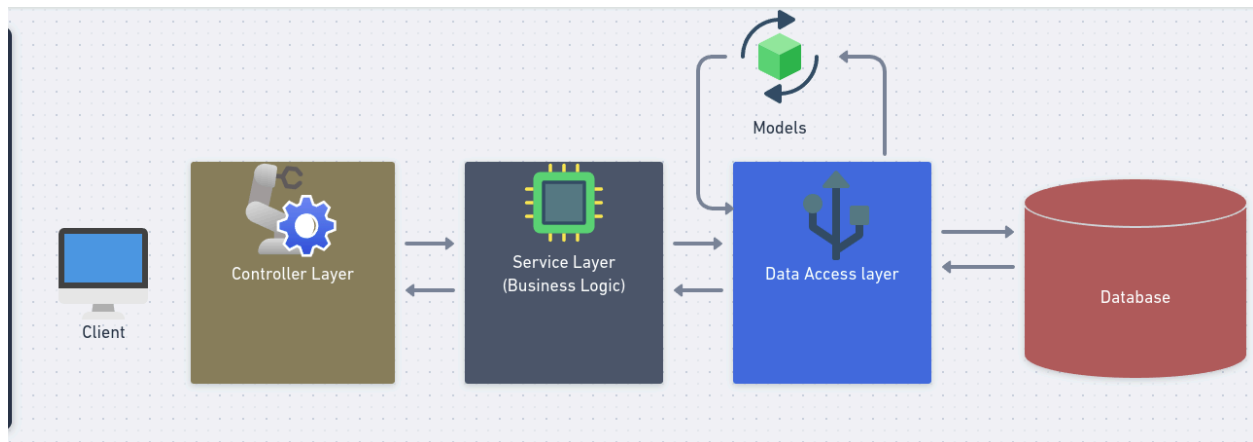
INTAKE_MANAGEMENT_PKG: Handles intake records, providing procedures for insert, delete, and update.

PROGRAM_MANAGEMENT_PKG: Manages program information with procedures for insert, delete, and update.

STUDENT_MANAGEMENT_PKG: Manages student data, including procedures for insert, delete, and update.

TRACK_MANAGEMENT_PKG: Manages track information, offering procedures for insert, delete, and update.

4.3. Application Implementation:



1. iti_simulation Package:

- This package houses the controllers classes and FXML files responsible for the user interface (UI) views.
- Following the Model-View-Controller (MVC) pattern, controllers manage user interactions and application logic, while FXML files define the layout and structure of UI components.
- Utilizing design patterns like the Observer pattern enhances the responsiveness and efficiency of table views, ensuring seamless updates upon data changes.
- The Adapter pattern is employed for views containing multiple tables, facilitating smooth handling and presentation of complex data structures.

2. Service Package:

- The Service package contains classes that bridge the gap between controllers and the data access layer, implementing business logic and handling flow control.
- Key responsibilities include email and password validation, data retrieval, constraint checking, and enforcing business rules.
- By encapsulating these operations within service classes, we promote modularity and maintainability while ensuring a clear separation of concerns.

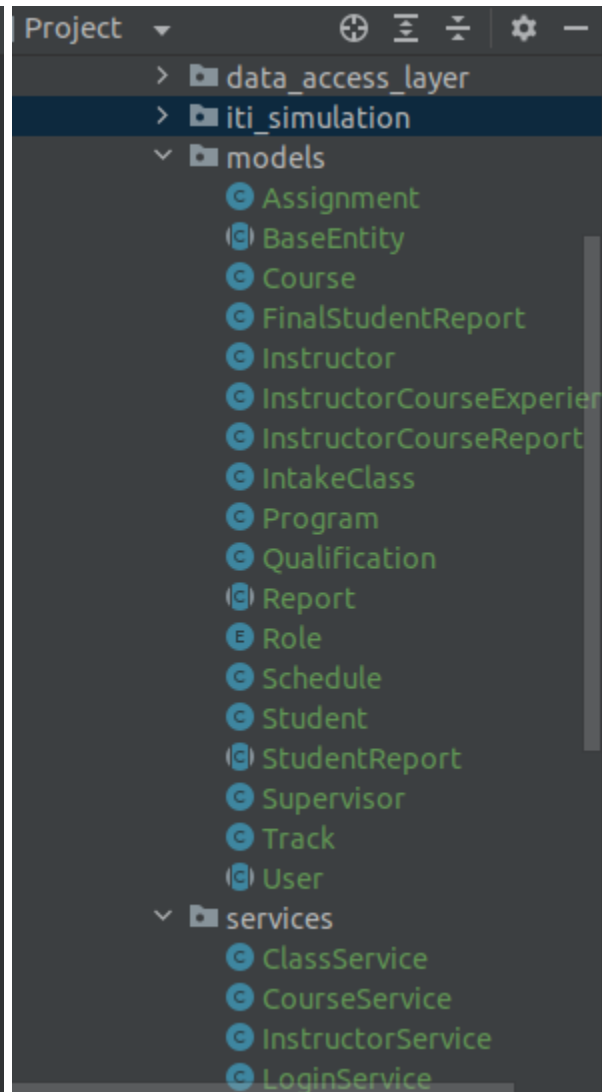
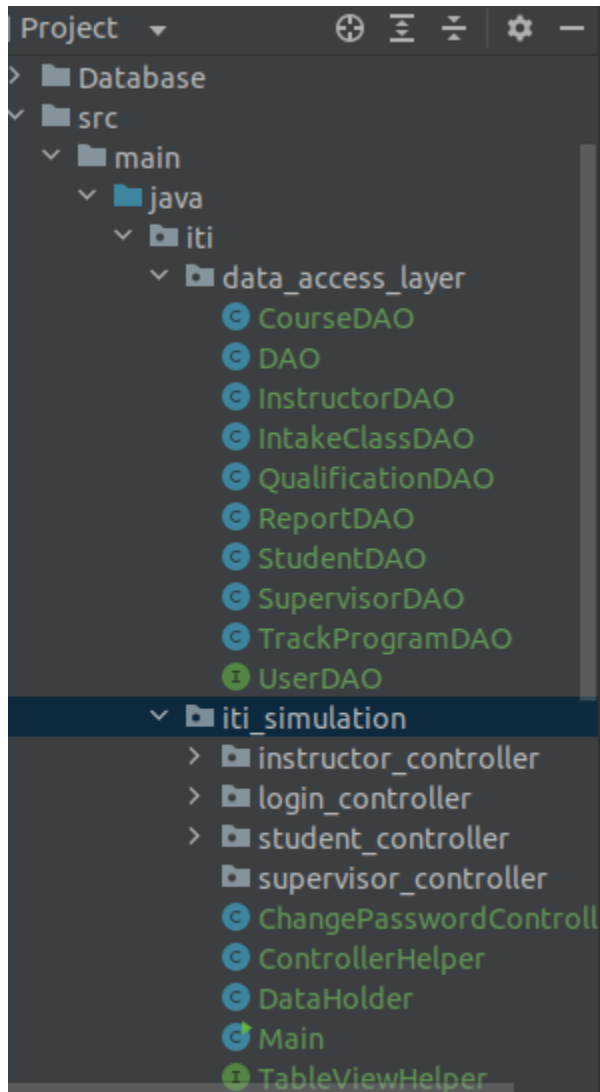
3. Data Access Layer:

- This package is dedicated to managing database connections and executing queries to retrieve or manipulate data.
- Classes within this layer encapsulate database interactions, abstracting the underlying implementation details.
- By adhering to this architectural design, we achieve a modular and scalable approach to database access, facilitating seamless integration with the rest of the application.

4. Model Package:

- The Model package contains classes representing transferable and shared objects used throughout the system.

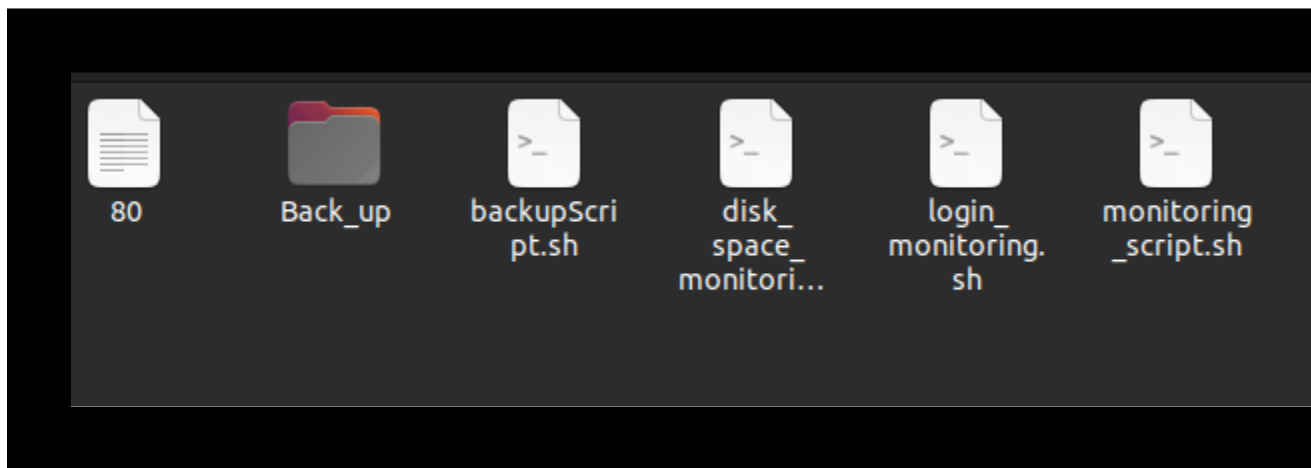
- These classes encapsulate data and behavior, serving as the backbone for communication and data flow within the application.
- By centralizing data management in the Model layer, we ensure consistency and integrity across different components of the system, promoting reusability and maintainability.



4.4. Monitoring and Backup Implementation:

These Bash scripts perform the following tasks:

1. Database Backup Script: Performs an Oracle database export using `expdp`, storing the backup file in a specified directory. It checks for successful completion and logs the result.
2. Disk Space Monitoring Script: Monitors disk space usage, logging alerts to a specified log file if disk space falls below a threshold (10%).
3. System Monitoring Script: Monitors CPU, memory, and disk usage thresholds. It logs alerts to a specified log file if any of these resources exceed their defined thresholds.



5. Further Improvement and Maintenance:

in the further improvement and maintenance phase I want to test normalization of my database to make it in an efficient normalized form and implement the supervisor view with functionalities.

6. References:

<https://www.youtube.com/watch?v=1YPT6VH256w>

<https://vertabelo.com/blog/database-design-management-system/>

<https://www.studocu.com/row/document/la%DB%81or-girizhn-ioniors%D9%B9i/database-systems/university-management-system-project-report-pdf/35226018>

<https://itsourcecode.com/uml/university-management-system-er-diagram-entity-relationships-diagram/>

<https://github.com/shivang2k/UniversityManagementSystem/tree/master>

<https://www.techprofree.com/database-design-and-implementation-tutorial-pdf/>

<https://www.studocu.com/in/document/galgotias-university/b-tech/university-management-system-database-management-system/40749655>

<https://www.javatpoint.com/er-diagram-for-the-university-management-system>

<https://github.com/topics/university-management-system>

<https://www.youtube.com/watch?v=CUR8K55JkYg>

<https://gist.github.com/jewelsea/4631319>

<https://www.edrawsoft.com/article/er-diagrams-for-university-database.html>

<https://vertabelo.com/blog/er-diagram-for-a-university-database/>

<https://slideplayer.com/slide/6087575/>