Software Requirements Specification

for

Tutor

Version 1.0 approved

Prepared by Vinod Warnakulasooriya

Team Kylix

2025 July 28

Contents

Revision History

Name	Date	Reason For Changes	Version
Vinod Warnakulasooriya	2025 July 28	Create Document	Version 1.0

1. Introduction

1.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for the Tutors application; a mobile and web-based solution designed for the Sri Lankan tuition industry. The app aims to automate the management of student attendance, class fee calculation, and reporting for tuition classes and institutes.
The target users include
Tutors:

- Create Account & Delete Account recover password.
- Add/remove their qualifications and experience.
- Add/Edit/ Remove their Bank Details to get fees.
- Add/Edit/ Remove class details.
- Add/Edit/ Remove students from their class.
- Add/Edit/ Remove tuition fee value per class. Per month per day or per Hour or create time duration.
- Add/ Remove Student Profile.
- Request to student join his class by student ID.
- Accept joining requests from students for joining class.
- Add/Edit/ Remove class timetable.
- mark attendance.
- Give Medals for students.
- Search Institute by Institute Number.
- Search Student by Student Number.
- calculated fees
- Mark printed notes give or not
- results published.
- Get Student Details by student QR code or Registration Number.
- calculating overall income at the end of month and ongoing live income.
- See their individual class, Own single class and institute classes.
- students count class wise.
- class wise income.
- Expect Income and Actual income.
- Income after cutting institute commission.
- Generate/Edit/ Remove QR code to share their profile.
- Request to institute to enter that institute.

- Generate invoices/bill to specific students or class or institute.
- Share generated invoices/bill to students or class or institute.
- Add/Edit/ Remove coupon code per class or institute.
- Send SMS to alert students when they pay fees.
- Send SMS if have any important announcement.
- Add Other products and prices (Papers, Tutes, T shirts etc…)
- Multi-Language Support.
- Send money Withdrawal alert to Institute.
- Accept or withdrawal money received alert.

Students/Parents:

- View Account & Delete Account recover password.
- Add/Edit/ Remove School.
- Add/Edit/ Remove Education qualifications.
- Search Tutor by tutor Number.
- Search Institute by Institute Number.
- Show or hide their marks and medals for others.
- Request for entering class.
- Accept class request.
- Follow tutor.
- Pay fees after Tutor or institute send invoice.
- See last month's details.
- See and download last month's bills.
- Get SMS alert when paying fee.
- See students' marks and medals.
- Multi-Language Support

Administrators:

- Create, Edit, and Delete Institute Accounts.
- Create, Edit, and Delete Tutor Accounts.
- Create, Edit, and Delete Student Accounts.
- Suspend or block any user (Tutor, Student, or Institute).
- Approve or reject Institute registration requests.
- View all Institutes and their details.
- Track Institute's tutors, classes, and students.
- Set commission percentage per institute.
- Add Kylix convenience fee for tutors and institutes.
- View and manage all registered tutors and students.
- Reset passwords for users if needed.
- View tutor qualification and verification documents (approve/reject).
- Resolve disputes between tutor and student.

- View overall income report (daily, monthly, yearly).
- Generate and download financial reports.
- Track commission per institute and deduct automatically.
- View payment history for all users.
- Approve or reject withdrawal requests from tutors.
- Monitor all classes created by tutors and institutes.
- Remove inappropriate content (class names, descriptions).
- Audit attendance and timetable records for compliance.
- View student enrollment trends.
- Total income vs expected income.
- Institute-wise income, Tutor-wise income, Class-wise income.
- Student activity reports (attendance, payment, engagement).
- Monitor top-performing tutors and classes.
- Manage system-wide settings (currency, SMS gateway, email notifications).
- Manage payment gateways.
- Manage promotional banners and advertisements in the platform.
- Multi-Language Support
- Send money Withdrawal alert to Tutor or Institute.
- Send Bill amount alert to institute or tutor.

Institute:

- Create, Edit, and Delete Institute profile.
- Add/Remove tutors to their institute.
- Add/Remove students to their institute.
- Approve tutor join requests.
- Approve student join requests.
- Create, Edit, and Delete Subjects.
- Create, Edit, and Delete Classes under subjects.
- Assign tutors to classes/subjects.
- Add students to classes.
- Manage class timetable (add/edit/remove).
- View students enrolled class-wise.
- Set class fees (per day, per month, per hour, or custom duration).
- Generate and share invoices/bills with students.
- Track pending and completed payments.
- Apply discount/coupon codes for students.
- Track income report for all classes under the institute.
- Calculate income after tutor payment and institute commission.
- Download and share monthly reports.
- Send SMS/email announcements to students/tutors.

- Send reminders for fee payment deadlines.
- Share important updates or schedules.
- Monitor student marks, medals, attendance.
- Track tutor performance (income, class ratings).
- Handle student complaints or tutor feedback.
- Sell extra products like books, paper, T-shirts.
- Generate QR codes for institute profile sharing.
- Send money Withdrawal alert to Tutor.
- Multi-Language Support

This document serves as a guideline for developers, testers, and stakeholders to understand the project scope and requirements.

1.2 Scope

- The Tutorz application will provide an integrated platform to:
- Mark and manage student attendance digitally.
- Automatically calculated fees based on attendance.
- Generate professional reports (PDF/Excel) for attendance and fee summaries.
- Provide real-time updates to students and parents through notifications.
- Enable tutors and administrators to manage classes and student data.

The system will include:

- Mobile Application: For tutors, students, and parents (Android & iOS).
- Web Application: For administrators and tutors for detailed management and reporting.
- Backend System: Built using .NET Core (three-tier architecture) with RESTful API for communication.
- Database: A relational database (MS SQL recommended for structured data and reporting).

The application will operate in an online environment with cloud-based hosting and will support scalability for thousands of students and multiple tuition centers.

1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirements Specification
- UI: User Interface
- API: Application Programming Interface
- CRUD: Create, Read, Update, Delete
- JWT: JSON Web Token (used for authentication)
- MS SQL: Microsoft SQL Server

1.4 References

- IEEE 830 Standard for SRS
- Microsoft .NET Core Documentation
- React.js Official Documentation

1.5 Overview

This document describes the functional requirements, non-functional requirements, system features, and interfaces of the Tutorz application.
It also includes details about the overall system architecture, database design, and constraints to ensure proper implementation and maintainability of the solution.

2. Overall Description

2.1 Product Perspective

The Tutorz application is a centralized, cloud-based system that consists of:

- Mobile App (for Students, Parents, Tutors, Institutes)
- Web App (for Students, Parents, Tutors, Institutes and Admin)
- Backend API (built with .NET Core, three-tier architecture)
- Database (MS SQL for structured data, scalable for multi-institute usage)
- Frontend React Vite with Atom architecture.
-

The system is designed as a **multi-tenant platform**, meaning multiple institutes can register and manage their own tutors and students independently under a single application.

The major components include:

- Institute Management: Institutes can register, create subjects, classes, add tutors, assign tutors to subjects, and add students to classes.
- Tutor Management: Tutors can mark attendance, manage their classes, and track fees.
- Student Portal: Students and parents can view attendance, fee details, and reports.
- Admin Dashboard: Super Admin monitors institute and ensures system integrity.

2.2 Product Functions

The core functionality of the Tutorz system includes:

Institute Functions

- Register and manage institute profile
- Add tutors and students
- Create subjects and classes
- Assign tutors to subjects
- Enroll students into classes
- Monitor attendance and fee reports

Tutor Functions

- Mark daily attendance for students
- View student list and class schedule
- Generate fee calculations based on attendance
- Download reports (PDF/Excel)

Student/Parent Functions

- View attendance history
- View fee details and payment status

- Receive notifications about classes and dues

Admin Functions

- Manage and approve institutes
- Monitor overall usage and analytics.

2.3 User Classes and Characteristics

User Role	Description	Technical Level
Admin	System owner, manages all institutes and configurations	High
Institute	Manages tutors, students, classes, and subjects	Medium
Tutor	Marks attendance, checks fees, generates reports	Low
Student/Parent	Views attendance, payments, and reports	Low

2.4 Operating Environment

Web App:

- Runs on modern browsers (Chrome, Edge, Firefox).

Mobile App:

- Android (API Level 24+)
- iOS (13+)

Backend:

- ASP.NET Core, REST API

Database:

- MS SQL Server hosted on Azure or AWS (Not Decide)

2.5 Design and Implementation Constraints

Programming Languages:

- Frontend: React (Atomic Design)
- Backend: .NET Core (C#)

Database:

- MS SQL recommended for structured and relational data

Authentication:

- JWT tokens for secure login

Reports:

- Generate PDFs using iTextSharp or similar library

2.6 Assumptions and Dependencies

- Internet connectivity is required for all features.
- Users must have valid email/phone for registration.
- Institute must register before tutors and students can join.
- Notifications depend on push notification services (Firebase Cloud Messaging or Azure Notification Hub).

3. System Features (Functional Requirements)

3.1 Tutor Features

3.1.1 Create Account

Description:

- Allow a tutor to register and create an account in the system using personal and contact details.

Inputs:

- Full Name
- Email
- Phone Number
- Password
- Institute ID (Optional if joining later)

Outputs:

- Success message
- Tutor Profile created in the system

Processing:

- Validate input data (email uniqueness, password strength)
- Hash password and store in database
- Generate unique Tutor ID
- Send confirmation email/SMS

3.1.2 Delete Account

Description:

- Tutor can delete their accounts permanently from the system.

Inputs:

- Tutor ID
- Authentication token
- Confirmation (Yes/No)

Outputs:

- Account deleted message
- Tutor data archived (or hard deleted based on policy)

Processing:

- Authenticate tutor
- Remove data or mark as inactive in DB
- Trigger notification to institute and admin

3.1.3 Recover Password

Description:

- Tutors can reset passwords using OTP/email link.

Inputs:
- Email/Phone

Outputs:
- OTP or reset link

Processing:
- Verify account
- Send OTP or reset link
- Allow setting new password

3.1.4 Add/Remove Qualifications & Experience

Description:
- Add or remove academic qualifications.

Inputs:
- Qualification name
- Year
- Experience details

Outputs:
- Updated profile

Processing:
- Validate tutor ID
- Insert or delete record in DB

3.1.5 Add/Edit/Remove Bank Details

Description:
- Manage payment bank details.

Inputs:
- Bank Name
- Account Number
- IFSC/SWIFT

Outputs:
- Success message

Processing:
- Validate input
- Encrypt sensitive info
- Store in DB

3.1.6 Add/Edit/Remove Class Details

Description:
- Create or manage classes.

Inputs:
- Class Name
- Subject
- Schedule

Outputs:

- Updated class list

Processing:

- Validate input
- Insert/update/delete in DB

3.1.7 Add/Edit/Remove Students from Class

Description:

- Manually add or remove students from a class.

Inputs:

- Student ID
- Class ID

Outputs:

- Updated student-class mapping

Processing:

- Validate IDs
- Update mapping table

3.1.8 Add/Edit/Remove Tuition Fee

Description:

- Set or update fee structure.

Inputs:

- Class ID
- Fee amount
- Fee type (per month/day/hour)

Outputs:

- Fee structure updated

Processing:

- Validate input
- Update DB

3.1.9 Add/Remove Student Profile

Description:

- Tutors can create or delete student profiles.

Inputs:

- Student Name
- Contact details

Outputs:

- Student profile added/removed

Processing:

- Insert or delete in DB

3.1.10 Request Student to Join Class

Description:

- Tutor can invite a student to join a class using Student ID.

Inputs:

- Student ID
- Class ID

Outputs:

- Request sent

Processing:

- Validate IDs
- Insert request in DB
- Notify student

3.1.11 Accept Joining Requests

Description:

- Tutor accepts student's join request.

Inputs:

- Request ID

Outputs:

- Students added to class

Processing:

- Validate request
- Update mapping

3.1.12 Add/Edit/Remove Class Timetable

Description:

- Manage class schedules.

Inputs:

- Class ID
- Day, Time

Outputs:

- Updated timetable

Processing:

- Validate schedule
- Update DB

3.1.13 Mark Attendance

Description:

- Tutor marks daily attendance.

Inputs:

- Class ID
- Student IDs with Present/Absent

Outputs:

- Attendance record saved

Processing:

- Insert attendance

- Update fee calculation

3.1.14 Give Medals to Students

Description:
- Award virtual medals for performance.

Inputs:
- Student ID
- Medal type

Outputs:
- Medal assigned

Processing:
- Validate IDs
- Update rewards table

3.1.15 Search Institute by Number

Description:
- Find an institute using institute ID.

Inputs:
- Institute ID

Outputs:
- Institute details

Processing:
- Fetch data from DB

3.1.16 Search Student by Number

Description:
- Find student details by ID.

Inputs:
- Student ID

Outputs:
- Student details

Processing:
- Fetch from DB

3.1.17 Calculate Fees

Description:
- The system calculates fees based on attendance or fixed rate.

Inputs:
- Attendance records
- Fee structure

Outputs:
- Fee amount

Processing:
- Apply formula

- Update payment records

3.1.18 Mark Printed Notes Given or Not

Description:
- Track whether notes were given.

Inputs:
- Student ID
- Status (Given/Not)

Outputs:
- Updated notes status

Processing:
- Update in DB

3.1.19 Results Published

Description:
- Tutor publishes student results.

Inputs:
- Student ID
- Marks

Outputs:
- Result visible to student/parent

Processing:
- Validate input
- Insert in DB

3.1.20 Get Student Details by QR Code or Reg Number

Description:
- Scan QR or enter Reg No to view student info.

Inputs:
- QR code / Reg Number

Outputs:
- Student profile details

Processing:
- Decode QR or search in DB

3.1.21 Calculate Overall Income

Description:
- Show monthly and live income.

Inputs:
- Fee payments
- Commission rate

Outputs:
- Expected vs Actual income report

Processing:

- Sum payments
- Deduct commissions

3.1.22 View Class Income & Student Count

Description:
- Class-level income and student data.

Inputs:
- Class ID

Outputs:
- Student count, Income amount

Processing:
- Aggregate DB records

3.1.23 Generate/Edit/Remove QR Code

Description:
- Tutor generates QR for profile sharing.

Inputs:
- Tutor ID

Outputs:
- QR code

Processing:
- Generate QR using library
- Store in DB

3.1.24 Request to Join Institute

Description:
- Tutor requests to join an institute.

Inputs:
- Institute ID

Outputs:
- Request sent

Processing:
- Validate IDs
- Insert request record

3.1.25 Generate Invoice

Description:
- Generate invoices for student or class.

Inputs:
- Student/Class ID
- Amount

Outputs:
- Invoice PDF

Processing:

- Fetch data
- Create PDF

3.1.26 Share Invoice

Description:
- Send invoice via SMS or email.

Inputs:
- Contact info

Outputs:
- Invoice sent

Processing:
- Use email/SMS API

3.1.27 Add/Edit/Remove Coupon Code

Description:
- Manage discount codes.

Inputs:
- Coupon code
- Discount rate

Outputs:
- Coupon active/inactive

Processing:
- Insert/update in DB

3.1.28 Send SMS Alerts

Description:
- Send alerts for fees or announcements.

Inputs:
- Message text
- Student list

Outputs:
- SMS sent

Processing:
- Use SMS gateway

3.1.29 Add Other Products

Description:
- Tutor sells products like t-shirts or notes.

Inputs:
- Product name
- Price

Outputs:
- Added to catalog

Processing:

- Insert into DB

3.1.30 Multi-Language Support

Description:
- UI in multiple languages (Sinhala, Tamil, English).

Inputs:
- Selected language

Outputs:
- UI text in selected language

Processing:
- Use localization files

3.1.31 Withdrawal Alert

Description:
- Notify institute when tutor requests withdrawal.

Inputs:
- Amount

Outputs:
- Alert sent

Processing:
- Trigger notification

3.1.32 Accept Withdrawal Alert

Description:
Tutor acknowledges withdrawal confirmation.
Inputs:
- Confirmation Yes/No

Outputs:
- Status updated

Processing:
- Update transaction record

3.2 Student/Parent Features

3.2.1 View Account

Description:
- Student/Parent can view their account details including personal info, classes, and fee status.

Inputs:
- Student ID / Parent ID
- Authentication Token

Outputs:
- Account details displayed (Name, Contact, Classes, Fee info)

Processing:

- Authenticate user
- Fetch profile from DB
- Render data in UI

3.2.2 Delete Account

Description:
- Student/Parent can delete their accounts from the system.

Inputs:
- Student ID
- Confirmation (Yes/No)

Outputs:
- Account deleted message

Processing:
- Authenticate user
- Mark account as inactive or deleted from DB
- Remove from class mappings

3.2.3 Recover Password

Description:
- Student/Parent can reset password using OTP/email link.

Inputs:
- Email/Phone

Outputs:
- OTP or password reset link

Processing:
- Validate account
- Generate OTP or reset token
- Send via email/SMS

3.2.4 Add/Edit/Remove School

Description:
- Manage school details in profile.

Inputs:
- School Name
- Address
- Class Grade

Outputs:
- Updated profile

Processing:
- Authenticate user
- Insert/update/delete school details in DB

3.2.5 Add/Edit/Remove Education Qualifications

Description:

- Add or update academic qualifications.

Inputs:

- Qualification Name
- Year of Completion

Outputs:

- Updated profile

Processing:

- Validate data
- Insert/update/delete in DB

3.2.6 Search Tutor by Tutor Number

Description:

- Find a tutor using their unique Tutor ID.

Inputs:

- Tutor ID

Outputs:

- Tutor profile details

Processing:

- Search tutor in DB
- Display details

3.2.7 Search Institute by Institute Number

Description:

- Find an institute by its unique ID.

Inputs:

- Institute ID

Outputs:

- Institute details

Processing:

- Fetch from DB
- Show in UI

3.2.8 Show or Hide Marks and Medals

Description:

- Students can choose to make marks and medals public or private.

Inputs:

- Student ID
- Visibility status (Show/Hide)

Outputs:

- Updated visibility setting

Processing:

- Update flag in DB

3.2.9 Request for Entering Class

Description:
- Send a join request to a class.

Inputs:
- Class ID
- Student ID

Outputs:
- Request sent confirmation

Processing:
- Insert join request in DB
- Notify tutor

3.2.10 Accept Class Request

Description:
- Students accept tutor's invitation to join a class.

Inputs:
- Request ID
- Student confirmation

Outputs:
- Students added to class

Processing:
- Validate request
- Update mapping table

3.2.11 Follow Tutor

Description:
- Students can follow a tutor to receive updates.

Inputs:
- Tutor ID
- Student ID

Outputs:
- Follow status updated

Processing:
- Insert record in follow table

3.2.12 Pay Fees After Invoice

Description:
- Students can pay fees via the app after receiving an invoice.

Inputs:
- Invoice ID
- Payment details (Card/Bank/Online)

Outputs:
- Payment success/failure message
- Updated fee status

Processing:

- Validate invoice
- Process payment via gateway
- Update transaction record

3.2.13 See Last Month's Details

Description:
- Students can view last month's attendance and fee details.

Inputs:
- Student ID
- Month

Outputs:
- Attendance & Fee summary

Processing:
- Fetch from DB using filters

3.2.14 See and Download Last Month's Bills

Description:
- Download invoices of previous months.

Inputs:
- Student ID
- Month

Outputs:
- PDF invoice file

Processing:
- Generate PDF
- Provide download link

3.2.15 Get SMS Alert When Paying Fee

Description:
- Send SMS after successful payment.

Inputs:
- Phone number
- Payment amount

Outputs:
- SMS confirmation

Processing:
- Use SMS API after successful payment

3.2.16 See Students' Marks and Medals

Description:
- View academic performance and awards.

Inputs:
- Student ID

Outputs:

- Marks and medals list

Processing:

- Fetch performance data from DB

3.2.17 Multi-Language Support

Description:

- Allow app in Sinhala, Tamil, and English.

Inputs:

- Selected language

Outputs:

- UI text updates

Processing:

- Load localization files for selected language

3.3 Institute Features

3.3.1 Create Institute Account

Description:

- Institute can register on the platform by providing official details and login credentials.

Inputs:

- Institute Name
- Address
- Contact Number
- Email
- Password

Outputs:

- Institute profile created
- Confirmation message

Processing:

- Validate input details
- Hash and store password in DB
- Generate unique Institute ID
- Send confirmation email/SMS

3.3.2 Delete Institute Account

Description:
Institute can delete its account permanently.
Inputs:

- Institute ID
- Authentication Token
- Confirmation (Yes/No)

Outputs:

- Account deleted message

Processing:

- Authenticate institute
- Set account as inactive (or delete based on policy)
- Notify tutors and students linked to the institute

3.3.3 Recover Password

Description:
Institute can reset password via OTP or email link.
Inputs:
- Email/Phone

Outputs:
- OTP or password reset link

Processing:
- Validate institute account exists
- Send OTP or reset link
- Allow new password setting

3.3.4 Add Tutors

Description:
Institute can add tutors to its profile.
Inputs:
- Tutor Name
- Contact details
- Qualifications

Outputs:
- Tutor added successfully

Processing:
- Validate tutor data
- Insert tutor record linked to institute in DB

3.3.5 Remove Tutors

Description:
Institute can remove tutors from its system.
Inputs:
- Tutor ID

Outputs:
- Tutor removed confirmation

Processing:
- Validate tutor exists in institute
- Remove or deactivate tutor record

3.3.6 Add Students

Description:
Institute can add new students.
Inputs:

- Student Name
- Contact details
- Grade

Outputs:

- Student added successfully

Processing:

- Insert student details linked to institute in DB

3.3.7 Remove Students

Description:
Institute can remove students from the system.
Inputs:

- Student ID

Outputs:

- Student removed

Processing:

- Validate student exists
- Remove or deactivate student record

3.3.8 Create Subjects

Description:
Institute can create subjects taught at the institute.
Inputs:

- Subject Name

Outputs:

- Subject added successfully

Processing:

- Insert subject in DB linked to institute

3.3.8 Assign Tutor to Subject

Description:
Institute can assign a tutor to a specific subject.
Inputs:

- Tutor ID
- Subject ID

Outputs:

- Assignment successful

Processing:

- Validate IDs
- Insert into mapping table (Tutor-Subject)

3.3.9 Create Classes

Description:
Institute can create classes for subjects.

Inputs:
- Class Name
- Subject ID
- Schedule (Day/Time)

Outputs:
- Class created successfully

Processing:
- Validate subject exists
- Insert into class table

3.3.10 Add Students to Class

Description:
Institute can enroll students into classes.
Inputs:
- Student ID
- Class ID

Outputs:
- Students added to class

Processing:
- Validate IDs
- Insert into class-student mapping table

3.3.11 Manage Timetable

Description:
Institute can add, edit, or delete class schedules.
Inputs:
- Class ID
- Day/Time

Outputs:
- Updated timetable

Processing:
- Validate schedule
- Update in DB

3.3.12 View Attendance Reports

Description:
Institute can view attendance reports for all classes.
Inputs:
- Class ID
- Date Range

Outputs:
- Attendance summary

Processing:
- Fetch records from DB
- Aggregate attendance data

3.3.13 View Fee Reports

Description:
Institute can view income and pending fee details.
Inputs:
- Class ID
- Month

Outputs:
- Fee collection summary

Processing:
- Fetch payment data
- Calculating totals

3.3.14 Calculate Commission

Description:
System calculates institute commission and tutor income split.
Inputs:
- Fee collected
- Commission percentage

Outputs:
- Commission report

Processing:
- Apply commission formula
- Update financial records

3.3.15 Approve Tutor Join Requests

Description:
Institute can approve tutor requests to join the institute.
Inputs:
- Tutor ID
- Request ID

Outputs:
- Approval status updated

Processing:
- Validate request
- Update tutor-institute mapping

3.3.16 Generate Reports (PDF/Excel)

Description:
Institute can download financial and attendance reports.
Inputs:
- Report type
- Date range

Outputs:

- Downloadable report file

Processing:

- Fetch relevant data
- Format into PDF/Excel
- Provide download link

3.3.17 Multi-Language Support

Description:
Provide language options (Sinhala, Tamil, English).
Inputs:

- Language selection

Outputs:

- UI updates

Processing:

- Load localization files based on selection

3.4 Administrator Features

3.4.1 Create Institute Account

Description:

- Admin can manually create an institute account on behalf of the institute.

Inputs:

- Institute Name
- Address
- Contact Number
- Email
- Password

Outputs:

- Institute account created successfully
- Institute ID generated

Processing:

- Validate details
- Store in database with unique Institute ID
- Send login credentials via email/SMS

3.4.2 Edit Institute Account

Description:

- Admin can modify institute account details.

Inputs:

- Institute ID
- Updated fields (name, contact, address, etc.)

Outputs:

- Update confirmation message

Processing:

- Fetch institute details

- Update provided fields
- Save to DB and notify institute

3.4.3 Delete Institute Account

Description:
- Admin can delete an institute account from the system.

Inputs:
- Institute ID
- Confirmation (Yes/No)

Outputs:
- Deletion confirmation message

Processing:
- Check institute exists
- Delete or mark as inactive in DB
- Notify all linked tutors/students

3.4.4 Create Tutor Account

Description:
- Admin can create a tutor profile manually.

Inputs:
- Tutor Name
- Contact Info
- Email
- Password

Outputs:
- Tutor account created

Processing:
- Validate details
- Insert into DB
- Send login credentials to tutor

3.4.5 Edit Tutor Account

Description:
- Admin can update tutor details.

Inputs:
- Tutor ID
- Updated data

Outputs:
- Update success message

Processing:
- Retrieve tutor record
- Update in DB

3.4.6 Delete Tutor Account

Description:
- Admin can remove tutor accounts.

Inputs:
- Tutor ID
- Confirmation

Outputs:
- Tutor account removed

Processing:
- Validate tutor exists
- Remove or deactivate in DB

3.4.7 Create Student Account

Description:
- Admin can manually add student accounts.

Inputs:
- Student Name
- Email/Phone
- Password

Outputs:
- Student account created

Processing:
- Validate input
- Store in DB
- Send credentials

3.4.8 Edit Student Account

Description:
- Admin can modify student details.

Inputs:
- Student ID
- Updated fields

Outputs:
- Update success message

Processing:
- Fetch student details
- Update DB

3.4.9 Delete Student Account

Description
- Admin can remove student accounts.

Inputs:
- Student ID
- Confirmation

Outputs:
- Account removed

Processing:
- Validate existence
- Remove or deactivate in DB

3.4.10 Suspend or Block User

Description:
- Admin can temporarily suspend or block tutors, students, or institutes.

Inputs:
- User ID
- Reason
- Duration

Outputs:
- Suspension confirmation

Processing:
- Update account status in DB
- Notify user

3.4.11 Approve/Reject Institute Registration

Description:
- Admin reviews institute registration requests.

Inputs:
- Institute ID
- Approval decision

Outputs:
- Approval/Rejection notification

Processing:
- Check documents and details
- Update status in DB

3.4.12 View All Institutes

Description:
- Admin can list all registered institutes with details.

Inputs:
- Filter options

Outputs:
- List of institutes

Processing:
- Fetch institute data from DB
- Apply filters

4.13 Track Institute's Tutors, Classes, and Students

Description:
- Admin can see tutors, classes, and students linked to each institute.

Inputs:
- Institute ID

Outputs:
- Detailed linked data

Processing:
- Fetch relational records from DB

4.14 Set Commission Percentage

Description:
- Admin can set commission rates per institute.

Inputs:
- Institute ID
- Commission %

Outputs:
- Commission rate updated

Processing:
- Store new rate in DB

4.15 Add Kylix Convenience Fee

Description:
- Admin can set a platform service fee for tutors and institutes.

Inputs:
- Fee type
- Amount/Percentage

Outputs:
- Fee updated

Processing:
- Store settings in configuration table

4.16 View & Manage Registered Tutors/Students

Description:
- Admin can search, view, and manage all tutor/student accounts.

Inputs:
- Search filters

Outputs:
- Search results

Processing:
- Fetch from DB with filters

4.17 Reset Password

Description:
- Admin can reset passwords for any user.

Inputs:
- User ID
- New password

Outputs:
- Password reset confirmation

Processing:
- Hash password and update DB
- Notify user

4.18 Approve/Reject Tutor Qualifications

Description:
- Admin reviews qualification documents.

Inputs:
- Tutor ID
- Decision

Outputs:
- Approval/Rejection message

Processing:
- Verify document authenticity
- Update tutor status

4.19 Resolve Disputes

Description:
- Admin mediates disputes between tutors and students.

Inputs:
- Dispute ID
- Resolution decision

Outputs:
- Dispute closed

Processing:
- Review evidence
- Record decision

4.20 View Overall Income Report
Description:
Admin can see total income (daily, monthly, yearly).
Inputs:
Date range
Outputs:
Income report
Processing:
Aggregate payment records

4.21 Generate & Download Financial Reports
Description:
Admin can generate PDF/Excel reports.
Inputs:
Report type
Date range
Outputs:
Downloadable report file
Processing:
Fetch and format data

4.22 Track Commission Per Institute
Description:
Admin can view commission earnings from each institute.
Inputs:
Date range
Outputs:
Commission report
Processing:
Calculate commission from payment data

4.23 View Payment History
Description:
Admin can view all payments made by any user.
Inputs:

User ID (optional)
Outputs:
Payment records
Processing:
Fetch from DB

4.24 Approve/Reject Withdrawal Requests
Description:
Admin handles tutor/institute withdrawal requests.
Inputs:
Request ID
Decision
Outputs:
Request status updated
Processing:
Validate balance
Update payment status

4.25 Monitor All Classes
Description:
Admin can monitor every class on the platform.
Inputs:
Filters (institute/tutor)
Outputs:
Class list
Processing:
Fetch from DB

4.26 Remove Inappropriate Content
Description:
Admin can remove offensive class names/descriptions.
Inputs:
Class ID
Outputs:
Content updated
Processing:
Validate complaint
Update DB

4.27 Audit Attendance & Timetable
Description:
Admin ensures compliance in records.
Inputs:
Class ID
Date range
Outputs:
Compliance report
Processing:
Fetch and review records

4.28 View Student Enrollment Trends
Description:
Admin can view enrollment growth over time.
Inputs:

Date range
Outputs:
Trend graph/report
Processing:
Aggregate enrollment data

4.29 Total Income vs Expected Income
Description:
Admin compares actual income with projected income.
Inputs:
Date range
Outputs:
Comparison report
Processing:
Compare actual DB records vs forecasts

4.30 Institute-wise, Tutor-wise, Class-wise Income
Description:
Admin can see income split by entities.
Inputs:
Filters
Outputs:
Detailed income breakdown
Processing:
Aggregate payment records by category

4.31 Student Activity Reports
Description:
Admin sees attendance, payment, and engagement per student.
Inputs:
Student ID
Outputs:
Detailed activity report
Processing:
Pull multi-source data from DB

4.32 Monitor Top Performing Tutors/Classes
Description:
Admin sees the best performers.
Inputs:
Ranking criteria
Outputs:
Leaderboard
Processing:
Sort based on performance metrics

4.33 Manage System-wide Settings
Description:
Admin can change system settings (currency, SMS, email, etc.).
Inputs:
Setting type
New value
Outputs:
Settings updated

Processing:
Save in system config table

4.34 Manage Payment Gateways
Description:
Admin can add, edit, or remove payment gateways.
Inputs:
Gateway name
API keys
Outputs:
Gateway updated
Processing:
Store details in DB

4.35 Manage Banners/Ads
Description:
Admin can manage promotional banners on the platform.
Inputs:
Banner image/text
Outputs:
Banner displayed
Processing:
Upload file
Store in DB

4.36 Multi-Language Support
Description:
Admin can configure languages for the platform.
Inputs:
Language selection
Outputs:
UI updates
Processing:
Load translation files

4.37 Send Withdrawal Alerts
Description:
Admin can send notifications to tutors/institutes about withdrawals.
Inputs:
User ID
Message content
Outputs:
SMS/Email sent
Processing:
Fetch user contact info
Send via gateway

4.38 Send Bill Amount Alerts
Description:
Admin can send bill payment alerts.
Inputs:
User ID
Bill amount
Outputs:

Alert sent
Processing:
Retrieve contact info
Send SMS/email

4. Non-Functional Requirements

5.1 Performance Requirements

- The system should support **at least 10,000 concurrent users** (students, tutors, and institutes).
- All user-facing operations (like login, marking attendance, or viewing reports) must complete within **2 seconds** under normal network conditions.
- Invoice and PDF report generation should complete within **5 seconds**.
- The mobile app should work smoothly even on **low-end devices (2GB RAM, quad-core processors)**.
- API responses must be optimized for bandwidth, with **payloads under 100KB for most requests**.

5.2 Scalability

- The system must scale horizontally to accommodate **growth of users, classes, and institutes**.
- Backend services should be **containerized (Docker/Kubernetes)** for easy scaling.
- Database should support **sharding and replication** to handle high traffic.
- Load balancers will be implemented to manage traffic between services.

5.3 Security

- All sensitive data (passwords, bank details) must be encrypted with **AES-256 encryption** and passwords hashed using **bcrypt or Argon2**.
- All endpoints will require **JWT-based authentication and role-based authorization**.
- Multi-factor authentication (MFA) support for tutors and institutes.
- API communication must use **HTTPS/TLS 1.2+** encryption.
- Admins can audit and monitor suspicious activities with logs stored for **minimum 90 days**.
- Compliance with **Sri Lankan Data Protection Act** for user privacy.

5.4 Usability

- Mobile app must be user-friendly, designed for students, parents, and tutors with minimal tech knowledge.
- All important features (attendance marking, payment, class management) must be accessible in **≤3 clicks/taps**.
- Provide **Sinhala, Tamil, and English language support**.
- Dark mode and accessibility options (font size adjustment, color contrast).
- Simple onboarding process with in-app tutorials.

5.5 Availability & Reliability

- System uptime should be **99.9% annually**.
- Automatic failover support in case of server failure.
- Cloud-based deployment (Azure/AWS) with redundancy for databases and services.
- Critical services (payment processing, attendance marking) must be **fault-tolerant** and retry in case of temporary failures.

5.6 Maintainability

- Code should follow **clean architecture principles** (three-tier backend, atom design frontend).
- Comprehensive API documentation and automated tests must cover **at least 80% of code**.
- Modular design to allow easy addition of new features.
- CI/CD pipelines for automated deployment and rollback in case of errors.

5.7 Portability

- The mobile app will run on **Android (6.0+) and iOS (13+)**.
- The web app will be compatible with **Chrome, Firefox, Safari, Edge (latest 2 versions)**.
- Backend services will be containerized for easy migration to any cloud provider.

5.8 Data Integrity

- Data consistency will be ensured using **ACID-compliant transactions** in the database.

- Regular **daily backups** with the ability to restore within **30 minutes** of data loss.
- Audit trails for every data change (attendance, payments, class edits).

- ## 5.9 Compliance
  - The system must comply with:
    - **Sri Lanka's ICTA guidelines** for digital platforms.
    - **PCI DSS** for payment security.
    - GDPR-like data privacy standards (right to access, delete data).

- ## 5.10 Disaster Recovery
  - Backup servers must be maintained in a **separate data center** or availability zone.
  - Full disaster recovery plan with **RPO (Recovery Point Objective) = 24 hours** and **RTO (Recovery Time Objective) = 2 hours**.

- ## 5.11 Logging & Monitoring
  - Centralized logging system (e.g., ELK stack, Azure Monitor).
  - Real-time monitoring of critical metrics: API response time, error rates, payment success rates.
  - Automatic alerts sent to admins for anomalies (high error rate, server downtime).

5. System Architecture

5.1 High-Level Architecture Diagram

5.1.1 Web Application Detailed High-Level Architecture (Tutor App)

1. Frontend (React App)

- Built with **React (Vite)** for fast build times and **TailwindCSS** for styling.
- Uses **Atom Design System**:
    - **Atoms**: Buttons, Inputs, Icons
    - **Molecules**: LoginForm, ClassCard
    - **Organisms**: Tables, Modals
    - **Pages**: Dashboard, Attendance, Payments
- **State management**: Redux Toolkit or Zustand for global state.
- **Routing**: React Router v6.
- **API Calls**: Axios for REST APIs.

---

2. API Gateway & Security Layer

- Nginx or Azure Front Door for **routing, SSL, and load balancing**.
- **JWT authentication** and **role-based access control** for Tutor/Admin/Student.
- **CSRF and CORS protection** for secure browser communication.
- Request throttling for DDoS protection.

---

3. Backend Services (ASP.NET Core 9)

- Built with a **Three-Tier Architecture**:
    - Controllers → Services → Repositories
- Each module is **role-focused**:
    - Tutors: Attendance, Invoices, Class Management
    - Institutes: Tutor Management, Reporting

- Admin: Moderation, Analytics, Withdrawals

- Business rules (commission logic, fee tracking) live here.

---

4. Database Layer

- **MS SQL Server** for relational data (Tutor, Student, Class, Payments).
- **Redis Cache** for high-performance lookups (tokens, profile data).
- Entity Framework Core for ORM + stored procedures for heavy reports.

---

5. Integrations

- **Payment Gateway** (Stripe, PayHere for Sri Lanka).
- **SMS Gateway** (Dialog SMS or Twilio).
- **Email Notifications** (SendGrid or AWS SES).
- **Cloud Storage** for invoices, QR codes, class materials.

6. Recommended System Architecture

🔷 Backend Architecture: **Clean Architecture + Three-Tier + Microservice-Ready**

[Presentation Layer (API)]

↓

[Application Layer (Business Logic, DTOs, Services)]

↓

[Domain Layer (Entities, Core Rules)]

↓

[Infrastructure Layer (EF Core, SQL Server, External APIs)]

🔥 **Why this is best:**

- **Three-Tier Approach** keeps **API, business logic, and data access separated**:
  - Easy to test and maintain.

- **Clean Architecture** allows for **future microservices**:
    - Payments, Notifications, and Reports can later become separate services.
- **Entity Framework Core**: Perfect for managing your evolving schema (Code-First).
- **Domain-Driven Design (DDD) inspired**: Entities are first-class, easier to grow.
- **Security**: JWT-based authentication, role-based authorization (Tutor, Student, Admin).
- **API Gateway Ready**: If the app grows large, APIs can be split per module.

🤗 Backend Tech Stack

Layer	**Tech**
Presentation Layer (API)	ASP.NET Core Web API 9
Business Layer	C#, MediatR, FluentValidation
Domain Layer	Plain C# Entities, Value Objects
Data Access	EF Core, LINQ, Stored Procedures
Database	SQL Server 2022
Authentication	JWT, OAuth2 (future)
External Services	Twilio (SMS), Stripe/PayPal (Payments), Email SMTP
CI/CD	Azure DevOps or GitHub Actions

🔷 **Frontend Architecture: SPA with Atomic Design**

[React + Vite SPA]

↓ (API calls)

[REST API Gateway (ASP.NET Core)]

↓

[Business Layer (Services, DTOs)]

↓

[Database (SQL Server)]

🔥 **Why this is best:**

- **Single-Page App (SPA)**: Fast, smooth user experience.
- **Atomic Design**: Organizes UI in **Atoms → Molecules → Organisms → Templates → Pages** (great for your huge feature list).
- **Component Reusability**: One AttendanceTable component can be reused for Tutors, Students, Admins.
- **State Management**: Redux Toolkit or React Query for caching and performance.
- **Tailwind CSS**: Fast styling for web + easy theming.

🌐 Frontend Tech Stack

Feature	**Tech**
Framework	React + Vite
Styling	Tailwind CSS
State Management	Redux Toolkit + RTK Query OR React Query
Routing	React Router
Form Handling	React Hook Form + Yup
Charts	Recharts / Chart.js
Notifications	Toastify / Sonner
Multi-Language Support	i18next

🔑 Why This is BEST for Your Project:

Requirement	**Why This Architecture Works**
Large Feature Set	Modularized layers (Clean Architecture) → easy to add features.
Role-Based Access	Easy role handling in API Middleware (Admin, Tutor, Student).
Future Mobile App	Same backend serves mobile & web → Reuse APIs.
Scaling	Can easily move to microservices (Payments, Notifications).
Performance	EF Core with caching, Stored Procedures for heavy reporting.

Requirement	Why This Architecture Works
Security	JWT, RBAC, and .NET's built-in Identity support.
UI Scalability	Atomic Design → easy to add 40+ Tutor features without chaos.

7. 🗂 Frontend File Structure (React + Vite + Tailwind)

```
tutor-web-app/
├── public/
│   ├── favicon.ico
│   ├── logo.png
│   └── manifest.json
│
├── src/
│   ├── assets/
│   │   ├── images/
│   │   │   └── logo.svg
│   │   └── styles/
│   │       └── global.css
│   │
│   ├── components/
│   │   ├── atoms/
│   │   │   ├── Button.jsx
│   │   │   ├── Input.jsx
│   │   │   ├── Select.jsx
│   │   │   ├── Label.jsx
│   │   │   └── Icon.jsx
│   │   │
│   │   ├── molecules/
│   │   │   ├── FormField.jsx
│   │   │   ├── ClassCard.jsx
│   │   │   └── InvoiceCard.jsx
│   │   │
│   │   ├── organisms/
│   │   │   ├── NavBar.jsx
│   │   │   ├── SideBar.jsx
│   │   │   ├── AttendanceTable.jsx
│   │   │   ├── StudentList.jsx
│   │   │   └── ReportsTable.jsx
│   │   │
│   │   └── templates/
│   │       ├── DashboardLayout.jsx
│   │       ├── AuthLayout.jsx
│   │       └── PageHeader.jsx
│   │
│   ├── pages/
```

```
        ├── auth/
        │   ├── LoginPage.jsx
        │   └── RegisterPage.jsx
        │
        ├── dashboard/
        │   ├── DashboardPage.jsx
        │   ├── ClassesPage.jsx
        │   ├── AttendancePage.jsx
        │   ├── PaymentsPage.jsx
        │   ├── ReportsPage.jsx
        │   └── SettingsPage.jsx
        │
        └── notfound/
            └── NotFoundPage.jsx
    ├── hooks/
    │   ├── useAuth.js
    │   ├── useApi.js
    │   └── useTheme.js
    │
    ├── context/
    │   └── AuthContext.jsx
    │
    ├── services/
    │   ├── api/
    │   │   ├── apiClient.js
    │   │   ├── tutorService.js
    │   │   ├── studentService.js
    │   │   ├── instituteService.js
    │   │   └── adminService.js
    │   │
    │   └── auth/
    │       └── authService.js
    │
    ├── routes/
    │   └── AppRoutes.jsx
    │
    ├── store/
    │   ├── index.js
    │   ├── authSlice.js
    │   ├── tutorSlice.js
    │   └── uiSlice.js
    │
    ├── utils/
    │   ├── constants.js
    │   ├── helpers.js
    │   └── validators.js
    │
    ├── App.jsx
    ├── main.jsx
    └── index.css
├── .env
├── package.json
```

```
├── vite.config.js
└── tailwind.config.js
```

- # 🔍 Structure Explanation

1. `public/`

- Static files like `favicon.ico`, app icons, and `manifest.json`.

---

2. `src/assets/`

- Contains `images/` and `global.css` styles.

---

3. `src/components/` (Atomic Design)

- **Atoms**: Smallest UI elements (Button, Input, Icon).
- **Molecules**: Combination of atoms (FormField, ClassCard).
- **Organisms**: Larger UI blocks (NavBar, SideBar, Tables).
- **Templates**: Layout components (DashboardLayout, AuthLayout).

---

4. `src/pages/`

- Divided by **feature areas**:
  - `auth/` (Login/Register)
  - `dashboard/` (Classes, Attendance, Payments)
  - `notfound/` (404 page).

---

5. `src/hooks/`

- Custom React hooks like `useAuth`, `useApi`.

---

6. `src/context/`

- React Context for global state (AuthContext, ThemeContext).

---

7. `src/services/api/`

- API clients for Tutor, Student, Institute, Admin.

- `apiClient.js` for Axios instance with interceptors.

---

8. `src/routes/`

- Centralized route configuration with `AppRoutes.jsx`.

---

9. `src/store/`

- Redux store slices: `authSlice`, `tutorSlice`, `uiSlice`.

---

10. `src/utils/`

- Utility functions, constants, and validators.

---

11. Root Files

- `App.jsx`: Root component.
- `main.jsx`: Entry point for React.
- `vite.config.js`: Vite configuration.
- `tailwind.config.js`: Tailwind setup.
- `.env`: Environment variables (API base URL, etc.).

8. 🏗️ Backend File Structure (ASP.NET Core 9 + Clean Architecture)

```
tutor-backend/
│
├── TutorApp.Api/                    # Presentation Layer (API Controllers)
│   ├── Controllers/
│   │   ├── AuthController.cs
│   │   ├── TutorController.cs
│   │   ├── StudentController.cs
│   │   ├── InstituteController.cs
│   │   ├── AdminController.cs
│   │   └── PaymentController.cs
│   │
│   ├── Filters/
│   │   └── ExceptionMiddleware.cs
```

```
    ├── Startup.cs
    └── Program.cs

├── TutorApp.Application/              # Business Logic Layer (Core Use Cases)
    ├── Interfaces/
    │   ├── IAuthService.cs
    │   ├── ITutorService.cs
    │   ├── IStudentService.cs
    │   ├── IInstituteService.cs
    │   └── IAdminService.cs
    │
    ├── Services/
    │   ├── AuthService.cs
    │   ├── TutorService.cs
    │   ├── StudentService.cs
    │   ├── InstituteService.cs
    │   └── AdminService.cs
    │
    ├── DTOs/
    │   ├── Auth/
    │   │   ├── LoginRequest.cs
    │   │   ├── RegisterRequest.cs
    │   │   └── AuthResponse.cs
    │   ├── Tutor/
    │   │   └── TutorDto.cs
    │   ├── Student/
    │   │   └── StudentDto.cs
    │   ├── Institute/
    │   │   └── InstituteDto.cs
    │   └── Admin/
    │       └── AdminDto.cs
    │
    ├── Mappings/
    │   └── AutoMapperProfiles.cs
    │
    ├── Validators/
    │   └── TutorValidator.cs
    │
    └── Exceptions/
        └── BusinessException.cs

├── TutorApp.Domain/                   # Domain Layer (Entities & Core Rules)
    ├── Entities/
    │   ├── User.cs
    │   ├── Tutor.cs
    │   ├── Student.cs
    │   ├── Institute.cs
    │   ├── Class.cs
    │   ├── Attendance.cs
    │   ├── Payment.cs
    │   └── Report.cs
    │
    ├── Enums/
```

```
    │   ├── UserRole.cs
    │   └── PaymentStatus.cs
    │
    └── ValueObjects/
        └── Money.cs

├── TutorApp.Infrastructure/              # Data Access Layer (DB + Repositories)
    ├── Data/
    │   ├── TutorAppDbContext.cs
    │   └── DbInitializer.cs
    │
    ├── Repositories/
    │   ├── GenericRepository.cs
    │   ├── TutorRepository.cs
    │   ├── StudentRepository.cs
    │   ├── InstituteRepository.cs
    │   └── PaymentRepository.cs
    │
    ├── Configurations/
    │   └── EntityConfigurations.cs
    │
    ├── Migrations/
    │   └── [Timestamp]_InitialMigration.cs
    │
    └── StoredProcedures/
        └── FeeCalculation.sql

├── TutorApp.Shared/                      # Shared Layer (Common Utilities)
    ├── Constants/
    │   └── AppConstants.cs
    │
    ├── Helpers/
    │   ├── JwtHelper.cs
    │   ├── PasswordHasher.cs
    │   └── FileStorageHelper.cs
    │
    ├── Middleware/
    │   └── LoggingMiddleware.cs
    │
    └── Notifications/
        ├── EmailService.cs
        ├── SmsService.cs
        └── NotificationTemplates.cs

├── tests/                                # Unit & Integration Tests
    ├── TutorApp.UnitTests/
    │   └── TutorServiceTests.cs
    └── TutorApp.IntegrationTests/
        └── AuthControllerTests.cs

├── .env                                  # Environment variables
├── appsettings.json                      # Application configuration
├── appsettings.Development.json
├── docker-compose.yml                    # Container orchestration
```

```
├── Dockerfile                  # Build file for backend container
└── TutorApp.sln                # Solution file
```

🔍 Layer-by-Layer Explanation

1. TutorApp.Api (Presentation Layer)

- Exposes APIs using ASP.NET Core Controllers.
- Handles HTTP requests, authentication, and response formatting.
- Global exception handling via middleware.

---

2. TutorApp.Application (Business Layer)

- Contains **business logic** and **service implementations**.
- Uses DTOs for communication between API and services.
- Contains validation and mapping logic.

---

3. TutorApp.Domain (Core Layer)

- Contains **Entities, Enums, and Value Objects**.
- Pure C# POCO classes (no DB dependencies).
- Defines system rules (e.g., Fee calculation formulas).

---

4. TutorApp.Infrastructure (Data Layer)

- Implements **EF Core DbContext** for database access.
- Contains Repositories and SQL scripts (Stored Procedures).
- Houses migrations for versioned database schema.

---

5. TutorApp.Shared (Utilities)

- Common utilities: JWT handling, file storage, constants.
- Notification services for SMS/Email.
- Middleware for logging and error handling.

---

6. Tests Folder

- Unit tests for services.
- Integration tests for controller endpoints.

9. 🗄️ Tutor App Database ER Diagram (High-Level)

1. Users Table
Stores **all users** (Admins, Tutors, Students, Institute Owners).

Column Name	Data Type	Key	Description
UserID	UNIQUEIDENTIFIER	PK	Unique ID for every user.
Name	NVARCHAR(100)		Full name of the user.
Email	NVARCHAR(150)	UNIQUE	Email address.
PhoneNumber	NVARCHAR(15)		Phone number.
PasswordHash	NVARCHAR(MAX)		Encrypted password.
Role	NVARCHAR(50)		Role: Admin, Tutor, Student, Institute.
CreatedAt	DATETIME		Account creation date.
IsActive	BIT		Status of account.

📎 **Relationships**:
- **1 → 1** with **Tutors**, **Students**, **Institutes** (Each user has exactly one profile type).

---

2. Institutes Table

Column Name	Data Type	Key	Description
InstituteID	UNIQUEIDENTIFIER	PK	Unique ID for the institute.
UserID	UNIQUEIDENTIFIER	FK	Links to Users table.
Name	NVARCHAR(150)		Institute name.
Address	NVARCHAR(255)		Institute address.
ContactNumber	NVARCHAR(15)		Main contact number.
CreatedAt	DATETIME		Creation date.

📎 **Relationships**:
- **1 (Institute) → * (Classes)** (An institute can have many classes).
- **1 (Institute) → * (Tutors)** (An institute can onboard multiple tutors).

Tutors Table

Column Name	Data Type	Key	Description
TutorID	UNIQUEIDENTIFIER	PK	Unique ID for tutor.
UserID	UNIQUEIDENTIFIER	FK	Links to Users table.
InstituteID	UNIQUEIDENTIFIER	FK	If tutor belongs to an institute.
Qualification	NVARCHAR(255)		Tutor's qualifications.
ExperienceYears	INT		Years of experience.
BankDetails	NVARCHAR(MAX)		Encrypted bank account details.
CreatedAt	DATETIME		Created timestamp.

📎 **Relationships**:
- **1 (Tutor) → * (Classes)** (A tutor can create many classes).

Students Table

Column Name	Data Type	Key	Description
StudentID	UNIQUEIDENTIFIER	PK	Unique ID for student.
UserID	UNIQUEIDENTIFIER	FK	Links to Users table.
School	NVARCHAR(150)		School name.
Grade	NVARCHAR(50)		Grade/class level.
ParentContact	NVARCHAR(15)		Parent's phone number.
CreatedAt	DATETIME		Created timestamp.

📎 **Relationships**:
- **1 (Student) → * (Attendance)** (A student has many attendance records).
- **1 (Student) → * (Payments)** (A student can make multiple payments).

Classes Table

Column Name	Data Type	Key	Description
ClassID	UNIQUEIDENTIFIER	PK	Unique ID for class.
TutorID	UNIQUEIDENTIFIER	FK	Class owner (tutor).
InstituteID	UNIQUEIDENTIFIER	FK	Institute offering the class.
Subject	NVARCHAR(150)		Subject name.
FeeType	NVARCHAR(50)		Monthly/Hourly/Per Class.
FeeAmount	DECIMAL(10,2)		Fee amount.
Schedule	NVARCHAR(MAX)		JSON or string for schedule.
CreatedAt	DATETIME		Created timestamp.

📎 **Relationships**:
- **1 (Class) → * (Attendance)**
- **1 (Class) → * (Payments)**
- **M:N (Class ↔ Student)** (via Attendance or Enrollment).

Attendance Table

Column Name	Data Type	Key	Description
AttendanceID	UNIQUEIDENTIFIER	PK	Unique attendance record.
ClassID	UNIQUEIDENTIFIER	FK	Class attended.
StudentID	UNIQUEIDENTIFIER	FK	Student present/absent.
Date	DATE		Attendance date.
Status	NVARCHAR(10)		Present/Absent.

📎 **Relationships**:
- **1 (Class) → * (Attendance)**
- **1 (Student) → * (Attendance)**

Payments Table

Column Name	Data Type	Key	Description
PaymentID	UNIQUEIDENTIFIER	PK	Unique payment record.
StudentID	UNIQUEIDENTIFIER	FK	Who made payment.
ClassID	UNIQUEIDENTIFIER	FK	Which class.

Column Name	Data Type	Key	Description
Amount	DECIMAL(10,2)		Amount paid.
PaymentDate	DATETIME		Payment date.
Status	NVARCHAR(20)		Paid/Pending/Failed.

---

8. Invoices Table

Column Name	Data Type	Key	Description
InvoiceID	UNIQUEIDENTIFIER	PK	Invoice identifier.
StudentID	UNIQUEIDENTIFIER	FK	For which student.
ClassID	UNIQUEIDENTIFIER	FK	Linked class.
Amount	DECIMAL(10,2)		Invoice amount.
DueDate	DATE		Due date.
Status	NVARCHAR(20)		Paid/Unpaid.

---

9. Coupons Table

Column Name	Data Type	Key	Description
CouponID	UNIQUEIDENTIFIER	PK	Coupon identifier.
Code	NVARCHAR(50)		Coupon code.
Discount	DECIMAL(5,2)		Discount % or value.
ExpiryDate	DATE		Expiry date.

---

10. Notifications Table

Column Name	Data Type	Key	Description
NotificationID	UNIQUEIDENTIFIER	PK	Notification ID.
UserID	UNIQUEIDENTIFIER	FK	Who receives notification.
Title	NVARCHAR(100)		Title.
Message	NVARCHAR(MAX)		Message content.
SentDate	DATETIME		When sent.

---

11. QR Codes Table

Column Name	Data Type	Key	Description
QRCodeID	UNIQUEIDENTIFIER	PK	QR code ID.
TutorID	UNIQUEIDENTIFIER	FK	Tutor profile QR code.
QRData	NVARCHAR(MAX)		QR encoded data.
CreatedDate	DATETIME		QR code creation date.

---

12. WithdrawalRequests Table

Column Name	Data Type	Key	Description
WithdrawalID	UNIQUEIDENTIFIER	PK	Request ID.
TutorID/InstID	UNIQUEIDENTIFIER	FK	Who is withdrawing.
Amount	DECIMAL(10,2)		Withdrawal amount.
Status	NVARCHAR(20)		Pending/Approved/Rejected.

---

13. SystemSettings Table

Column Name	Data Type	Key	Description
SettingID	UNIQUEIDENTIFIER	PK	Setting identifier.
SettingKey	NVARCHAR(100)		Key (e.g., CommissionRate).
SettingValue	NVARCHAR(255)		Value.

- 🔗 **Cardinality (Key Relationships)**
- **Users → Tutors/Students/Institutes**: **1 → 1** (One user has exactly one role profile).
- **Institutes → Tutors**: **1 → * ** (One institute has many tutors).
- **Tutors → Classes**: **1 → * ** (One tutor teaches many classes).
- **Classes → Students**: **M:N** (Many students can join many classes; tracked via Attendance).
- **Students → Attendance**: **1 → * ** (One student has multiple attendance records).
- **Classes → Attendance**: **1 → * ** (One class has many attendance records).
- **Students → Payments**: **1 → * ** (A student makes multiple payments).
- **Invoices → Payments**: **1 → * ** (Each invoice can have one or more payments).
- **Tutor/Institute → WithdrawalRequests**: **1 → * ** (They can have multiple requests).

👨 Core Entities

Table Name	Attribute Name	Data Type	Key/Constraint	Notes
Role	RoleId	INT/GUID	PK	
	Name	VARCHAR(50)	Unique	e.g., 'Admin', 'Tutor', 'Student'
	Description	VARCHAR(255)	Nullable	
	IsActive	BOOLEAN	Default True	
	CreatedDate	DATETIME	Non-Nullable	Audit field
	UpdatedDate	DATETIME	Non-Nullable	Audit field
---	---	---	---	---
User	UserId	GUID/INT	PK	
	Email	VARCHAR(100)	Unique, Non-Nullable	Login credential
	PhoneNumber	VARCHAR(20)	Unique, Nullable	
	PasswordHash	VARCHAR(255)	Non-Nullable	Stored securely
	PasswordSalt	VARCHAR(255)	Nullable	Used with hashing (if applicable)
	RoleId	INT/GUID	FK → Role.RoleId	Many-to-One → Role
	IsActive	BOOLEAN	Default True	
	CreatedDate	DATETIME	Non-Nullable	
	UpdatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Tutor	TutorId	GUID/INT	PK	

	UserId	GUID/INT	FK → User.UserId, Unique	One-to-One → User
	FullName	VARCHAR(150)	Non-Nullable	
	Bio	TEXT	Nullable	
	ExperienceYears	INT	Default 0	
	BankAccountNumber	VARCHAR(50)	Nullable	For payments/withdrawals
	BankName	VARCHAR(100)	Nullable	
	IsActive	BOOLEAN	Default True	
	CreatedDate	DATETIME	Non-Nullable	
	UpdatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Student	StudentId	GUID/INT	PK	
	UserId	GUID/INT	FK → User.UserId, Unique	One-to-One → User
	FullName	VARCHAR(150)	Non-Nullable	
	SchoolName	VARCHAR(100)	Nullable	
	Grade	VARCHAR(20)	Nullable	e.g., '10', 'A-Level'
	ParentContact	VARCHAR(20)	Nullable	
	IsActive	BOOLEAN	Default True	
	CreatedDate	DATETIME	Non-Nullable	
	UpdatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Institute	InstituteId	GUID/INT	PK	
	UserId	GUID/INT	FK → User.UserId, Unique	One-to-One → User
	InstituteName	VARCHAR(150)	Non-Nullable	
	Address	VARCHAR(255)	Nullable	
	ContactNumber	VARCHAR(20)	Nullable	
	Website	VARCHAR(255)	Nullable	
	IsActive	BOOLEAN	Default True	
	CreatedDate	DATETIME	Non-Nullable	
	UpdatedDate	DATETIME	Non-Nullable	

💰 Financial Entities

Table Name	Attribute Name	Data Type	Key/Constraint	Notes
Subject	SubjectId	INT/GUID	PK	
	Name	VARCHAR(100)	Unique	
	Description	TEXT	Nullable	
	CreatedDate	DATETIME	Non-Nullable	
	UpdatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Class	ClassId	GUID/INT	PK	
	TutorId	GUID/INT	FK → Tutor.TutorId	Many-to-One → Tutor
	InstituteId	GUID/INT	FK → Institute.InstituteId	Nullable (can be run by Tutor only)
	SubjectId	GUID/INT	FK → Subject.SubjectId	Many-to-One → Subject
	Title	VARCHAR(150)	Non-Nullable	
	Description	TEXT	Nullable	
	Schedule	VARCHAR(255)	Nullable	e.g., 'Mon/Wed/Fri'
	Capacity	INT	Nullable	Max number of students
	IsActive	BOOLEAN	Default True	
	CreatedDate	DATETIME	Non-Nullable	
	UpdatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Enrollment	EnrollmentId	GUID/INT	PK	
	ClassId	GUID/INT	FK → Class.ClassId	Part of Unique key: (ClassId,StudentId)
	StudentId	GUID/INT	FK → Student.StudentId	Part of Unique key: (ClassId,StudentId)
	EnrollmentDate	DATE	Non-Nullable	
	IsActive	BOOLEAN	Default True	
	CreatedDate	DATETIME	Non-Nullable	
	UpdatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Timetable	TimetableId	GUID/INT	PK	
	ClassId	GUID/INT	FK → Class.ClassId	Many-to-One → Class
	DayOfWeek	ENUM/INT	Non-Nullable	e.g., 1=Monday, 7=Sunday

	StartTime	TIME	Non-Nullable	
	EndTime	TIME	Non-Nullable	
	CreatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Attendance	AttendanceId	GUID/INT	PK	
	ClassId	GUID/INT	FK → Class.ClassId	Part of Unique key: (ClassId,StudentId,Date)
	StudentId	GUID/INT	FK → Student.StudentId	Part of Unique key: (ClassId,StudentId,Date)
	Date	DATE	Non-Nullable	
	Status	VARCHAR(10)	Non-Nullable	e.g., 'Present', 'Absent', 'Late'
	CreatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Result	ResultId	GUID/INT	PK	
	ClassId	GUID/INT	FK → Class.ClassId	Part of Unique key: (ClassId,StudentId,ExamName)
	StudentId	GUID/INT	FK → Student.StudentId	Part of Unique key: (ClassId,StudentId,ExamName)
	ExamName	VARCHAR(100)	Non-Nullable	e.g., 'Midterm', 'Final Exam'
	Marks	DECIMAL(5,2)	Nullable	
	Grade	VARCHAR(10)	Nullable	e.g., 'A+', 'Pass', 'Fail'
	ExamDate	DATE	Non-Nullable	
	CreatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Medal	MedalId	GUID/INT	PK	
	TutorId	GUID/INT	FK → Tutor.TutorId	Many-to-One → Tutor
	StudentId	GUID/INT	FK → Student.StudentId	Many-to-One → Student
	Title	VARCHAR(100)	Non-Nullable	e.g., 'Top Scorer', 'Most Improved'
	Description	TEXT	Nullable	
	AwardedDate	DATE	Non-Nullable	
	CreatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Qualification	QualificationId	GUID/INT	PK	
	TutorId	GUID/INT	FK →	Many-to-One → Tutor

			Tutor.TutorId	
	QualificationName	VARCHAR(150)	Non-Nullable	e.g., 'M.Sc. Physics'
	Institution	VARCHAR(150)	Non-Nullable	
	YearObtained	INT	Non-Nullable	
	CreatedDate	DATETIME	Non-Nullable	

⚙ System & Utility Entities

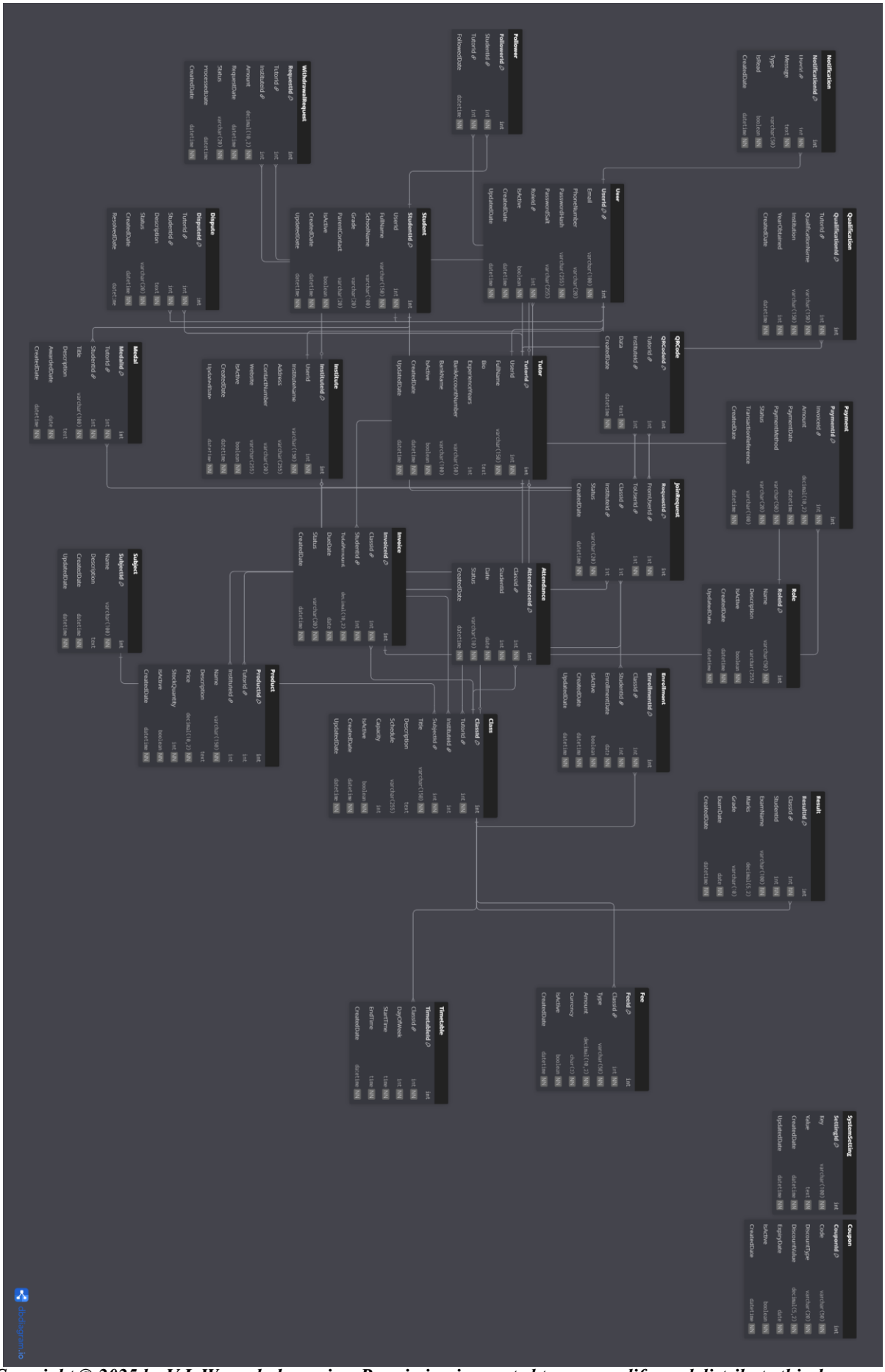
Table Name	Attribute Name	Data Type	Key/Constraint	Notes
JoinRequest	RequestId	GUID/INT	PK	
	FromUserId	GUID/INT	FK → User.UserId	The user initiating the request
	ToUserId	GUID/INT	FK → User.UserId	The user/owner receiving the request
	ClassId	GUID/INT	FK → Class.ClassId	Nullable (e.g., student joining a class)
	InstituteId	GUID/INT	FK → Institute.InstituteId	Nullable (e.g., tutor joining institute)
	Status	VARCHAR(20)	Non-Nullable	e.g., 'Pending', 'Accepted'
	CreatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
QRCode	QRCodeId	GUID/INT	PK	
	TutorId	GUID/INT	FK → Tutor.TutorId	Nullable
	InstituteId	GUID/INT	FK → Institute.InstituteId	Nullable
	Data	TEXT	Non-Nullable	The actual QR code payload (URL, encoded data, etc.)
	CreatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Notification	NotificationId	GUID/INT	PK	
	UserId	GUID/INT	FK → User.UserId	Many-to-One → User
	Message	TEXT	Non-Nullable	
	Type	VARCHAR(50)	Nullable	e.g., 'Payment', 'Announcement'
	IsRead	BOOLEAN	Default False	

	CreatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Dispute	DisputeId	GUID/INT	PK	
	TutorId	GUID/INT	FK → Tutor.TutorId	Many-to-One → Tutor
	StudentId	GUID/INT	FK → Student.StudentId	Many-to-One → Student
	Description	TEXT	Non-Nullable	
	Status	VARCHAR(20)	Non-Nullable	e.g., 'Open', 'Resolved', 'Closed'
	CreatedDate	DATETIME	Non-Nullable	
	ResolvedDate	DATETIME	Nullable	
---	---	---	---	---
SystemSetting	SettingId	GUID/INT	PK	
	Key	VARCHAR(100)	Unique, Non-Nullable	e.g., 'TaxRate', 'MaxFileSize'
	Value	TEXT	Non-Nullable	The stored value
	CreatedDate	DATETIME	Non-Nullable	
	UpdatedDate	DATETIME	Non-Nullable	
---	---	---	---	---
Follower	FollowerId	GUID/INT	PK	
	StudentId	GUID/INT	FK → Student.StudentId	Part of Unique key: (StudentId,TutorId)
	TutorId	GUID/INT	FK → Tutor.TutorId	Part of Unique key: (StudentId,TutorId)
	FollowedDate	DATETIME	Non-Nullable	

https://dbdiagram.io/d/Tutorz-68b9073d61a46d388e5c3338

https://dbdocs.io/vinuwa98/Tutor?table=User&schema=public&view=table_structure

Password for DBDocs – 123456



9.1 User Documentation

To ensure users can effectively and independently operate the Tutorz application, a comprehensive suite of user documentation will be developed and maintained. The documentation will be tailored to the specific needs of each user class: Tutors, Students/Parents, Institute Administrators, and System Administrators. All documentation will be available in English, Sinhala, and Tamil to align with the application's multi-language support feature.

- **1.6.1 User Manual for Tutors and Institutes**

This will be the primary reference guide for professional users who manage content and operations within the platform.

- **Purpose and Audience:** This manual is intended for Tutors and Institute staff who need detailed, step-by-step instructions on all administrative and management features of the application. It will cater to users with a low-to-medium technical proficiency.
- **Format and Delivery:** The manual will be available in two formats:
    - An online, searchable web-based help center accessible through the Tutorz website and web app.
    - A downloadable PDF version for offline reference.
- **Scope and Content:** The manual will provide exhaustive coverage of all features available to Tutors and Institutes, including but not limited to:
    - **Account Management:** Creating, editing, and deleting accounts ; password recovery ; and managing profile details like qualifications, experience, and bank information.
    - **Class and Student Management:** Creating and managing subjects and classes , managing class timetables , adding/removing tutors (for Institutes) , and enrolling/removing students from classes.
    - **Daily Operations:** Marking student attendance , tracking the distribution of printed notes , and publishing student results or awarding medals.
    - **Financial Management:** Setting tuition fees , generating and sharing invoices , applying coupon codes , tracking class-wise and overall income , calculating institute commission , and managing withdrawal requests.
    - **Reporting:** Generating and downloading attendance, fee, and income reports in PDF and Excel formats.

- **Communication:** Sending SMS alerts and announcements to students and parents.

- **1.6.2 Quick Start Guide for Students and Parents**

This guide will focus on simplifying the onboarding process for the largest user group.

- **Purpose and Audience:** This document is designed for Students and Parents, who are classified as having a low technical level. Its goal is to get them comfortable with the essential features of the app quickly.
- **Format and Delivery:**
  - A concise, one-page digital flyer available for download.
  - An interactive, skippable pop-up guide presented upon the user's first login to the mobile app.
- **Scope and Content:** The guide will cover only the core functionalities relevant to students and parents:
  - Registering a new account and logging in.
  - Searching for tutors, classes, and institutes.
  - Requesting to join a class and accepting invitations from a tutor.
  - Navigating the user dashboard to view attendance history, marks, and medals.
  - Viewing and downloading monthly bills.
  - Making a fee payment securely through the app.

- **1.6.3 Administrator's Guide**

This documentation is for the internal technical team responsible for maintaining the entire platform.

- **Purpose and Audience:** This guide is for the System Administrator, who has a high technical level and requires detailed knowledge of the backend system, configurations, and moderation tools.
- **Format and Delivery:** A secure, internal technical document accessible only to authorized personnel.
- **Scope and Content:** This guide will detail all administrator-level features, such as:
  - **User and Institute Management:** Manually creating, editing, suspending, or deleting any user or institute account.

- **Platform Configuration:** Setting and managing institute commission percentages , configuring the "Kylix convenience fee" , and managing system-wide settings like payment gateways, SMS providers, and promotional banners.
- **Monitoring and Auditing:** Tracking platform-wide income and generating financial reports , monitoring all classes and content for appropriateness , auditing attendance records , and viewing student enrollment trends.
- **Support and Moderation:** Approving institute registrations , verifying tutor qualifications , resolving disputes between users , and handling withdrawal requests.

- **1.6.4 In-App Contextual Help & FAQs**

To provide immediate, on-demand support, help will be integrated directly into the user interface.

- **Purpose and Audience:** To provide instant clarification to all users as they interact with the application, reducing the need to consult external documentation.
- **Format and Delivery:** Integrated directly within the web and mobile applications.
- **Scope and Content:**
  - **Interactive Tours:** Optional guided walkthroughs that introduce users to new features or complex workflows upon their first encounter.
  - **Tooltips:** Small info-icons placed next to specific fields or settings (e.g., "Fee Type," "Commission Rate") that provide a brief explanation when hovered over or tapped.
  - **FAQ Section:** A searchable repository of frequently asked questions and answers, covering common issues like "How do I reset my password?" or "Why has my payment failed?".

9.2 Operational Scenarios

The following scenarios describe the step-by-step interactions of different user classes with the Tutorz system to accomplish specific goals. These narratives are intended to provide a clear context for the functional requirements detailed in Section 3.

- **Scenario 1: A New Tutor Onboards and Creates a Class**
- **Objective:** A new tutor, Ms. Dias, wants to register on the platform, join an existing tuition institute, and set up her first class.
- **User Roles:** Tutor, Institute Administrator
- **Pre-conditions:** The "Success Academy" institute already has an approved account on the Tutorz platform.

Sequence of Events:

1. **Registration:** Ms. Dias downloads the Tutorz mobile app and successfully creates a new Tutor account by providing her name, email, and phone number.
2. **Profile Enhancement:** She logs in and navigates to her profile to add her academic qualifications and past teaching experience, making her profile more attractive to institutes and students.
3. **Joining an Institute:** Ms. Dias wants to affiliate with "Success Academy." She uses the search function to find the institute by its unique number and sends a request to join.
4. **Institute Approval:** The administrator for "Success Academy" logs into their web dashboard and sees a pending join request from Ms. Dias. The administrator reviews her qualifications and approves the request. Ms. Dias receives a confirmation notification.
5. **Class Creation:** Now affiliated with the institute, Ms. Dias creates her first class, "Grade 11 Chemistry." She enters the class details, sets the schedule, and defines the fee structure as a fixed monthly amount.
6. **Outcome:** The "Grade 11 Chemistry" class is now active and listed under both Ms. Dias's profile and the "Success Academy" institute, ready for students to enroll.

- **Scenario 2: A Student Enrolls and Gets Attendance Marked**
- **Objective:** A parent, Mr. Fernando, wants to enroll his son, Nimal, in Ms. Dias's new chemistry class. Ms. Dias will then mark Nimal's attendance using the app.
- **User Roles:** Student/Parent, Tutor
- **Pre-conditions:** Ms. Dias's class is active. Mr. Fernando and Nimal have a shared Student/Parent account.

Sequence of Events:

1. **Discovery:** Mr. Fernando logs into the app and searches for Ms. Dias by her tutor number, which he got from a flyer.
2. **Enrollment Request:** He views her profile, sees the "Grade 11 Chemistry" class, and sends a request for Nimal to join the class.
3. **Tutor Acceptance:** Ms. Dias receives a notification for the join request. She reviews Nimal's profile and accepts, officially enrolling him in her class.

4. **Attendance Marking:** On the first day of class, Ms. Dias opens her class list in the app. She marks Nimal as 'Present' with a single tap. For another student, she uses the app's camera to scan their QR code to instantly bring up their profile and mark their attendance.
5. **Parental View:** After the class, Mr. Fernando can log in and view Nimal's attendance record, confirming he was present for the class.
6. **Outcome:** Nimal is successfully enrolled, and his attendance is logged in the system. This attendance record is now available for fee calculation at the end of the month.

- **Scenario 3: End-of-Month Billing and Tutor Income Withdrawal**
- **Objective:** At the end of the month, Ms. Dias bills Mr. Fernando for Nimal's classes. After payment, she withdraws her earnings from the platform.
- **User Roles:** Tutor, Student/Parent, Institute Administrator
- **Pre-conditions:** Attendance for the month has been recorded.

Sequence of Events:

1. **Invoice Generation:** The Tutorz system automatically calculates the fee for Nimal based on the fixed monthly rate. Ms. Dias reviews this and generates an official invoice within the app. She then shares it directly with Mr. Fernando through the app's communication feature.
2. **Fee Payment:** Mr. Fernando receives a notification with the invoice. He logs in, views the invoice details, and pays the full amount using his credit card via the integrated payment gateway.
3. **Confirmations:** Upon successful payment, Mr. Fernando instantly receives an SMS receipt. Simultaneously, the system marks the invoice as 'Paid,' and the transaction is reflected in the income reports for both Ms. Dias and "Success Academy." The system automatically deducts the institute's commission from Ms. Dias's earnings.
4. **Withdrawal Request:** A week later, Ms. Dias decides to withdraw her accumulated earnings. She submits a withdrawal request to the institute through the app.
5. **Withdrawal Approval:** The "Success Academy" administrator receives the request, verifies the amount against their records, and approves the transaction.
6. **Outcome:** The funds are transferred to Ms. Dias's registered bank account. The entire financial cycle—billing, payment, commission splitting, and withdrawal—is completed and logged within the Tutorz platform.

9.3 Appendices

This section contains supplementary information that supports the main content of the SRS, including a glossary of terms and a description of external system interfaces.

Appendix A: Glossary

This glossary defines key terms, acronyms, and abbreviations used throughout the Tutorz Software Requirements Specification to ensure a common understanding among all project stakeholders.

Term / Acronym	Definition
Admin	A super-user with system-wide privileges to manage all institutes, tutors, students, and settings on the Tutorz platform.
API	Application Programming Interface. A set of rules and protocols that allows different software applications to communicate with each other.
Commission	The percentage of the tuition fee that the platform (Kylix) or an Institute takes from a tutor's earnings.
CRUD	An acronym for Create, Read, Update, and Delete, which are the four basic functions of persistent storage.
Enrollment	The action of a student officially joining a class, which can be initiated by the student, tutor, or institute.
Institute	An organization or tuition center that registers on the platform to manage its own group of tutors, students, and classes.
Invoice	A digital bill generated by a Tutor or Institute for a student, detailing the amount due for tuition fees or other products.
JWT	JSON Web Token. A compact, URL-safe means of representing claims to be transferred between two parties, used for user authentication and authorization.
Multi-tenancy	An architecture where a single instance of the software serves multiple distinct user groups (in this case, multiple institutes operate independently on the same platform).
OTP	One-Time Password. A temporary, secure code sent via SMS or email for user verification, typically during registration or password recovery.
QR Code	A machine-readable code consisting of an array of black and white squares, used for storing URLs or other information for reading by the camera on a smartphone. In this app, it's used for sharing user profiles and retrieving student details.
SRS	Software Requirements Specification. This document, which formally defines the purpose and requirements of the application.

Withdrawal	The process by which a tutor or institute requests to transfer their earnings from the Tutorz platform to their personal bank account.

Export to Sheets

- **Appendix B: System Interfaces**

This section describes the interfaces between the Tutorz application and other external systems.

- **B.1 Payment Gateway Interface**
- **Purpose:** To securely process online fee payments from students/parents. The system will integrate with a third-party payment gateway compliant with Sri Lankan financial regulations (e.g., PayHere) or an international standard (e.g., Stripe).
- **Interface:** The Tutorz backend will communicate with the payment gateway via a secure **RESTful API over HTTPS**.
- **Data Exchange:** The system will send transaction details (e.g., invoice ID, amount, currency) to the gateway and receive back a payment status (success, failure) and a transaction reference token. All sensitive card details will be handled directly by the gateway to ensure PCI DSS compliance.
- **B.2 SMS Gateway Interface**
- **Purpose:** To send transactional and informational SMS messages to users. This includes sending OTPs for verification, payment confirmation receipts, and important announcements.
- **Interface:** The system will connect to a third-party SMS provider (e.g., Twilio, Dialog SMS) using a **RESTful API**.
- **Data Exchange:** The backend will make API calls sending the recipient's phone number and the message content. It will receive a status indicating whether the message was successfully sent.
- **B.3 Email Notification Service**
- **Purpose:** To send automated emails for user account activities, such as welcome emails, password reset links, and detailed monthly invoice statements.
- **Interface:** The backend will integrate with an email service provider (e.g., SendGrid, AWS SES) via **SMTP or a RESTful API**.
- **Data Exchange:** The system will provide the recipient's email address, subject, and email body (often using pre-defined templates) to the service for delivery.

Project Plan: Tutorz Application

This plan is structured in logical phases, starting with a foundational setup, moving to a Minimum Viable Product (MVP) that can be tested by early users, and then incrementally adding more advanced features.

- **Phase 1: Foundation & Core Setup (Weeks 1-3)**

Goal: Establish the project's technical foundation, including development environments, core architecture, and user authentication, which is central to all user roles.

Milestone 1.1: Project Initialization & Environment Setup (Week 1)

- **Tasks:**
    1. **Project Management:** Set up a project board (e.g., Trello, Jira, GitHub Projects) to track tasks.
    2. **Version Control:** Initialize Git repositories for both the frontend (tutor-web-app) and backend (tutor-backend).
    3. **Backend:**
        - Set up the solution structure for the ASP.NET Core application following the Clean Architecture described in the SRS (Domain, Application, Infrastructure, API projects).
        - Configure the database connection (MS SQL Server).
        - Implement the initial database schema using Entity Framework Core (Code-First) for the Users and Roles tables.
    4. **Frontend:**
        - Initialize the React + Vite project.
        - Set up the folder structure as outlined in the SRS (components, pages, services, etc.).
        - Integrate Tailwind CSS for styling.

Milestone 1.2: Core User Authentication & Identity (Weeks 2-3)

- **Goal:** Implement the ability for users to register, log in, and manage their passwords. This is a critical foundation for all other features.
- **Tasks:**
    1. **Backend (API):**
        - Develop API endpoints for user registration (Create Account), login, and password recovery (via OTP/email).
        - Implement JWT-based authentication to secure endpoints.
        - Create the database tables: Users, Tutors, Students, Institutes to store profile information.
        - Implement role-based authorization middleware (Admin, Tutor, Student, Institute).
    2. **Frontend (UI):**
        - Create the UI pages: Login, Register, Forgot Password, Reset Password.
        - Implement API integration using axios for these authentication flows.
        - Set up state management (e.g., Redux Toolkit) to handle user sessions and tokens.
        - Create a basic protected routing mechanism to restrict access to authenticated users.

- **Phase 2: Minimum Viable Product (MVP) - The Tutor Experience (Weeks 4-8)**

Goal: Develop the core features that allow a single tutor to manage their classes and students independently. This creates a usable product for the primary user group as quickly as possible.

Milestone 2.1: Tutor Profile & Class Management (Weeks 4-5)

- **Goal:** Enable tutors to set up their professional profile and create classes.
- **Tasks:**
    1. **Backend (API):**

        - Create CRUD endpoints for managing Classes (Add/Edit/Remove Class Details).
        - Develop endpoints for Tutors to manage their profiles (Add/Remove Qualifications & Experience, Bank Details).
        - Develop endpoints to manage the class timetable.
    2. **Frontend (UI):**
        - Build the Tutor Dashboard layout.
        - Create the "My Profile" page where tutors can edit their details.
        - Develop the "Class Management" page to list, create, and edit classes and their schedules.

Milestone 2.2: Student Management & Attendance (Weeks 6-7)

- **Goal:** Implement the core daily function of the app: managing students within a class and marking attendance.
- **Tasks:**
    1. **Backend (API):**
        - Create endpoints for tutors to add, view, and remove students from a class. This includes creating student profiles manually.
        - Develop the endpoint to mark student attendance for a specific class and date.
        - Implement logic to handle class joining requests from students and acceptance by tutors.
        - Create the Attendance table in the database.
    2. **Frontend (UI):**
        - Build the "View Class" page, which lists all enrolled students.
        - Implement the UI for marking attendance (e.g., a list of students with "Present"/"Absent" toggles).
        - Create a simple UI for managing join requests.

Milestone 2.3: Basic Fee Calculation & Student View (Week 8)

- **Goal:** Close the loop by allowing fee calculation and giving students a way to view their information.
- **Tasks:**
    1. **Backend (API):**
        - Develop an API endpoint that calculates fees based on attendance records.
        - Create endpoints for students/parents to view their own attendance history.
        - Create the Payments and Invoices tables.
    2. **Frontend (UI):**
        - On the Tutor's dashboard, display calculated fees for each student.
        - Create a basic Student/Parent view page to display attendance history.

- **Phase 3: Expanding to Institutes & Administrators (Weeks 9-13)**

Goal: Build upon the MVP by adding functionality for Institutes to manage multiple tutors and for Administrators to oversee the entire platform.

Milestone 3.1: Institute Management Features (Weeks 9-10)

- **Goal:** Enable institutes to function as umbrellas for tutors and classes.
- **Tasks:**
    1. **Backend (API):**
        - Implement endpoints for Institute account creation and profile management.
        - Develop endpoints for institutes to add/remove tutors and students.
        - Create logic for tutors to request to join an institute and for institutes to approve these requests.
        - Develop endpoints for creating subjects and assigning tutors to them.
    2. **Frontend (UI):**
        - Build the Institute Administrator dashboard.
        - Create UI components for managing lists of tutors, students, subjects, and classes.

        - Implement the workflow for approving tutor join requests.

Milestone 3.2: Administrator (Super Admin) Dashboard (Weeks 11-12)

- **Goal:** Provide system-wide oversight and management capabilities to the platform owner.
- **Tasks:**
    1. **Backend (API):**
        - Develop Admin-only endpoints to Create, Edit, and Delete any Institute, Tutor, or Student account.
        - Implement endpoints to suspend or block users.
        - Create endpoints to view all institutes and track their tutors/classes.
        - Implement the logic to set institute-specific commission percentages.
    2. **Frontend (UI):**
        - Design and build the Super Admin dashboard.
        - Create views for managing all users and institutes on the platform.
        - Develop forms for setting system-wide configurations like fees and commissions.

Milestone 3.3: Deployment & Initial Testing (Week 13)

- **Goal:** Deploy the application to a staging environment for internal testing.
- **Tasks:**
    1. **DevOps:**
        - Set up a cloud hosting environment (Azure or AWS).
        - Configure CI/CD pipelines for automated deployments.
        - Deploy the backend API and the frontend application.
    2. **Testing:**
        - Conduct end-to-end testing of all implemented features.
        - Fix critical bugs found during the testing phase.

- **Phase 4: Financials, Reporting & Advanced Features (Weeks 14-18)**

Goal: Enhance the platform with robust financial transaction capabilities, reporting, and user experience improvements.

Milestone 4.1: Full Financial Cycle (Weeks 14-15)

- **Goal:** Automate the entire billing, payment, and withdrawal process.
- **Tasks:**
    1. **Backend (API):**
        - Integrate a payment gateway (e.g., PayHere) for fee payments.
        - Develop endpoints for tutors/institutes to generate and share invoices.
        - Implement the withdrawal request and approval workflow (Tutor -> Institute -> Admin).
        - Integrate an SMS gateway (e.g., Dialog, Twilio) to send payment alerts.
    2. **Frontend (UI):**
        - Build the UI for invoice generation and viewing.
        - Integrate the payment gateway's interface for students to pay fees.
        - Create the UI for tutors/institutes to request withdrawals and for admins to approve them.

Milestone 4.2: Reporting & Analytics (Weeks 16-17)

- **Goal:** Provide valuable insights to users through data reports.
- **Tasks:**
    1. **Backend (API):**
        - Develop functionality to generate and download PDF/Excel reports for attendance and fees.
        - Create API endpoints for income reports (overall, class-wise, tutor-wise).
        - Aggregate data for admin dashboards (enrollment trends, top tutors).
    2. **Frontend (UI):**
        - Implement a "Reports" section for tutors, institutes, and admins.
        - Integrate a charting library (e.g., Chart.js) to visualize data.

    - Add buttons to download reports in specified formats.

Milestone 4.3: User Experience Enhancements (Week 18)

- **Goal:** Refine the application with features that improve usability and engagement.
- **Tasks:**
    1. **Backend (API) & Frontend (UI):**
        - Implement QR code generation for profiles and scanning for student details.
        - Implement the "Give Medals" feature for tutors.
        - Add multi-language support (Sinhala, Tamil, English) using a library like i18next.
        - Implement search functionality (search for tutors, institutes by number).

- **Phase 5: Finalization & Launch (Weeks 19-20)**

Goal: Prepare the application for a public launch by ensuring it is secure, reliable, and well-documented.

Milestone 5.1: Security, Performance & Final Testing (Week 19)

- **Tasks:**
    1. **Security:** Conduct a security audit. Ensure all non-functional requirements like data encryption and role-based access are strictly enforced.
    2. **Performance:** Optimize database queries and API response times to meet performance targets (seconds).
    3. **Testing:** Perform comprehensive user acceptance testing (UAT) with a small group of real tutors and students. Fix any remaining bugs.

Milestone 5.2: Documentation & Launch (Week 20)

- **Tasks:**
    1. **Documentation:** Prepare the user manuals, quick start guides, and FAQs as outlined in the SRS.
    2. **Production Deployment:** Deploy the application to the production environment.
    3. **Go-Live:** Announce the launch and open registration to the public.
    4. **Monitoring:** Set up real-time monitoring and alerting to track system health and user activity post-launch.