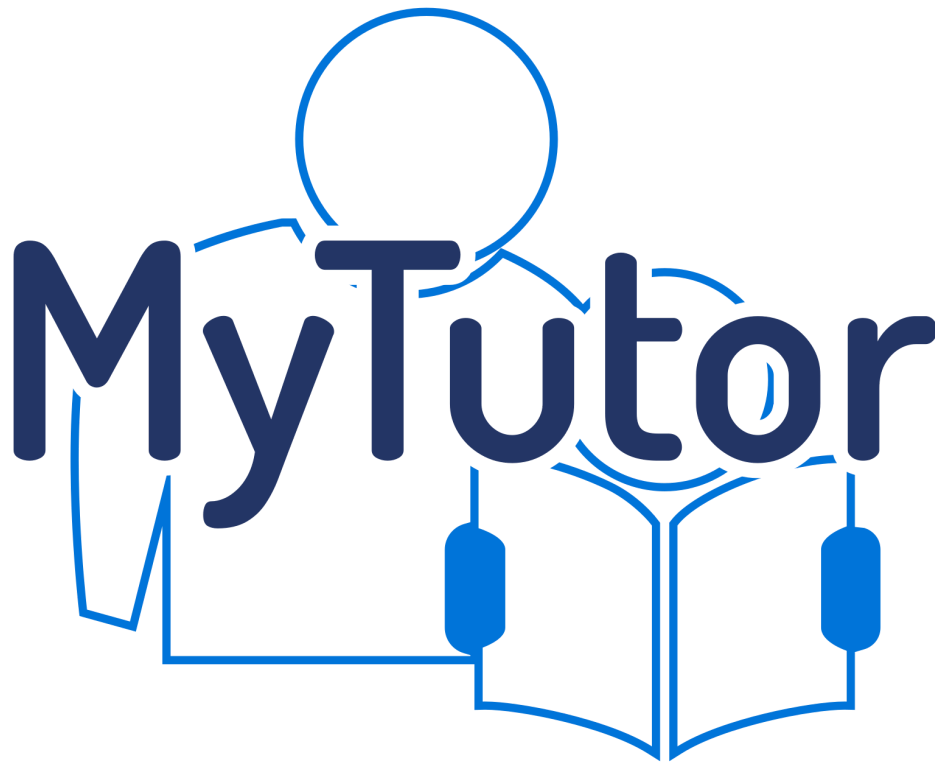


MyTutor Project Report



Julyan van der Westhuizen

Department of Computer Science

vwsjul003@myuct.ac.za

Cassandra Wallace

Department of Information Systems

wllcas004@myuct.ac.za

Ethan Wilson

Department of Computer Science

wlseth003@myuct.ac.za

Abstract

MyTutor is a comprehensive web-based application that aims to revolutionise the Tutor and Teaching Assistant (TA) management at the University of Cape Town (UCT). Catering to Administrators, Course Convenors, Lecturers, Tutors, TAs, and Students, the system facilitates seamless application, allocation, and monitoring of Tutors and TAs, significantly enhancing the efficiency of course management. It boasts a secure and user-friendly environment for managing the tutoring structures, allowing employees to effortlessly handle schedules. Key results of this project include the development of a modular and scalable system, guaranteeing adaptability to evolving requirements and smooth integration. Additional accomplishments include administrative user-management capabilities, robust user-validation, and an email notification system for password management. Overall, MyTutor's intuitive interface successfully simplifies tutor management tasks and provides students with a directly accessible and easy way to apply, alleviating the burden on Course Convenors. The system promises to enhance productivity across UCT's departments and faculties, ushering in a transformative era for tutor management. With its comprehensive design and user-centric approach, MyTutor is poised to become an essential tool in UCT's educational landscape.

1. Introduction

1.1. Project Context

Tutor Management is a vital aspect at academic institutions worldwide. At UCT, the Department of Computer Science employs nearly one hundred Tutors and TAs annually. The current manual process for managing them is complex and labour-intensive, causing challenges and frustrations for Course Convenors and impacting the quality of the tutoring experience for students. This burden can be alleviated through the introduction of an automated system. Thus, a desperate need for a Tutor Management System has been identified, one that will simplify procedures and enhance the overall tutoring experience.

1.2. Statement of Scope

In response to these challenges, our client tasked us with the production of MyTutor, a web-based application designed to simplify and improve the management, sign-up, and monitoring of Tutors and TAs. MyTutor allows Students to apply, Course Convenors to allocate staff, and Administrators to oversee tutoring activities, all while ensuring role-based security for a positive user experience.

1.3. System Objectives

MyTutor's primary objective is to streamline the Tutor and TA management process, alleviating the burden on Course Convenors and enhancing the quality and efficiency of tutoring within courses.

By providing a secure platform, it aims to:

- Allow users to securely sign-in to their accounts, accessing role-specific dashboards.
- Enable users to view course details, schedules, and apply for tutoring positions.
- Facilitate administrators in creating and managing user profiles & course information.
- Improve the overall efficiency and effectiveness of Tutor and TA management at UCT.

1.4. Software Engineering Methods

Our project team's approach to the development of the MyTutor software was to adopt an Agile Development Methodology, combining elements of traditional analysis with iterative development cycles akin to Scrum. This approach allowed us to efficiently tackle the project's complexities, whilst still ensuring adaptability throughout the development process.

We initiated the project with rigorous problem analysis and comprehensive documentation to establish a solid foundation. To provide a tangible vision of the product to our client, we presented two basic throw-away prototypes. The first emphasised the User Interface (UI), providing a broad, horizontal view of the application's flow and functionality. The second focused on the backend, demonstrating proof-of-concept for persistent database interaction during user sign-up. We opted to produce throw-away prototypes because we wanted to start afresh upon our first weekly Sprint, allowing us to maintain flexibility and incorporate client feedback effectively.

As we transitioned to full-scale implementation, we decided to adopt Scrum ideology by implementing week-long Sprints, where the project team prioritises a dense sprint-backlog to be realised during the week. This choice proved invaluable in adhering to our project plan and maintaining a rapid development pace. Week-long Sprint cycles allowed us to regularly review and adapt to evolving requirements and feedback from both each other and the client.

Our approach to solving the intricate challenges of Tutor Management led us to leverage various modern technologies, such as the Vaadin web-framework and Spring Boot, in our efforts to develop a robust and scalable system. Additionally, as requested by the client, we made sure to place significance on the use of object-oriented design principles.

This methodology and problem-solving approach enabled us to efficiently deliver a sophisticated solution that met our client's objectives, whilst maintaining a high-level of software quality and responsiveness to change. Subsequently, this report will delve deeper into MyTutor's architecture, functionalities, and outcomes, providing a comprehensive understanding of the system's future contribution to the academic landscape at UCT.

2. Requirements Captured

MyTutor is designed to manage Tutors and TAs for courses offered at UCT.

The system has the following functional requirements:

1. **User Authentication:** Users are able to securely sign-in to their accounts. This is implemented in the WelcomeView class, where users can enter their email and password to sign-in.
2. **User Management:** Administrators can create, edit, and delete user profiles.
3. **Course Management:** Administrators can create, edit, and delete course details.
4. **Application:** Students can apply to be considered to tutor or assist a course that they have previously completed.
5. **Acceptance:** Course Convenors (and Lecturers) can accept TAs to assist for a certain course. Course Convenors, Lecturers, and TAs can accept Tutors to tutor for a certain course.
6. **Course Viewing:** Users can view specific course details and tutoring schedules.
7. **Schedule Management:** Course Convenors, Lecturers, and TAs can create and edit a certain course's schedule, where they can add, modify, or delete tutoring sessions.

8. **Tutor Check-In:** Tutors can confirm their attendance for tutoring a tutoring session that they were signed up for.
9. **Statistics Viewing:** Course Convenors, Lecturers, and TAs can view the tutoring attendance sum for a selected Tutor, tutoring session, or for the course as a whole.

The system has the following non-functional requirements:

1. **Security:** It provides appropriate levels of security to protect user data and prevent unauthorised access. It implements various security measures, such as user authentication & authorisation, role-based access control, and secure transmission of sensitive information.
2. **Scalability:** The system should be scalable to accommodate potential growth in the number of users, courses, and tutoring sessions.
3. **Reliability:** The system must maintain high availability and data integrity, ensuring that users can access the system without disruptions and that data remains consistent.

Usability requirements focus on providing a user-friendly experience, including:

1. **Accessibility:** The system should be accessible to users with disabilities, complying with accessibility standards to ensure inclusivity.
2. **Intuitiveness:** The system should feature an intuitive and easy-to-navigate UI to ensure that all users, regardless of their roles, can efficiently perform their tasks.

To support these usability requirements, MyTutor makes use of Vaadin Flow web-framework for the UI. This allows the application to be very responsive and provide a maximally user-friendly experience. This is further achieved by ensuring that the UI is clean and simple, with clear labels and instructions. The system always provides feedback to the user, such as error messages and confirmation dialogs.

2.1. Use Case Narratives & Diagrams

Throughout the development process, our project team produced several analysis artefacts. Among these, we created comprehensive use case narratives and diagrams to outline the system's core functionalities, such as user sign-in, course viewing, application processes, and administrative actions. These narratives and diagrams served as blueprints for the development of MyTutor, ensuring that the system aligns with the intended requirements and stakeholder expectations.

“Sign-in”

As a user on MyTutor, I want to securely sign into my account to access my role-specific dashboard and fulfil my responsibilities.

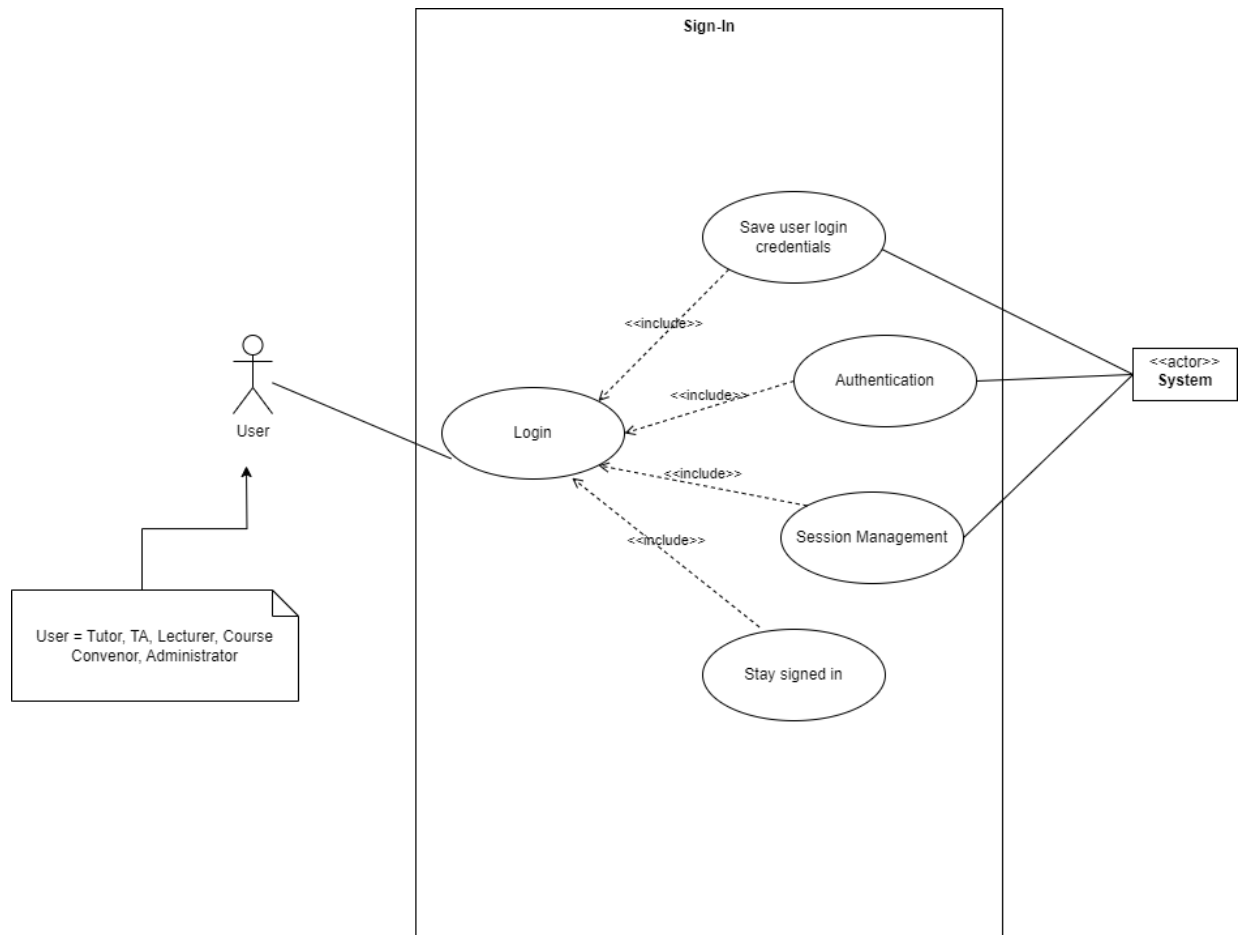


Diagram 1: Sign-In Use Case Diagram

“View a Course and it’s Schedule”

As a user on MyTutor, I want to be able to view a specific Course (that I either lecture, convene, assist, or tutor for), so I can read relevant information associated with the Course, such as its Schedule.

As a user on MyTutor, I want to be able to view the Schedule of a specific Course (that I either lecture, convene, assist, or tutor for), so that I can see all the TutoringSessions and plan my availability as required for my role.

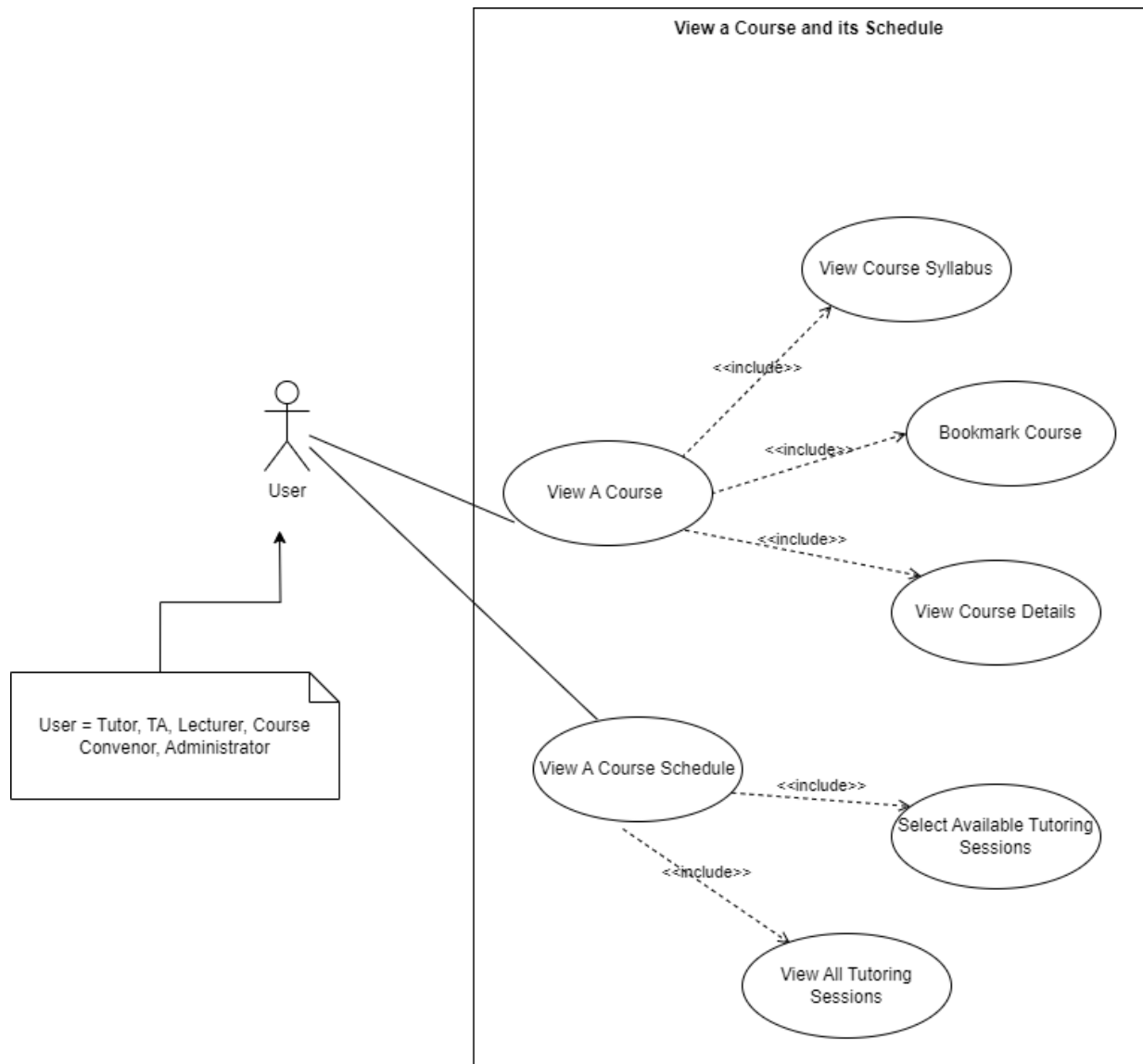


Diagram 2: Course and Schedule View Use Case Diagram

“Apply to Tutor a Course”

As a Tutor on MyTutor, I want to be able to apply to Tutor a specific Course, so that I can contribute to the Course's teaching activities.

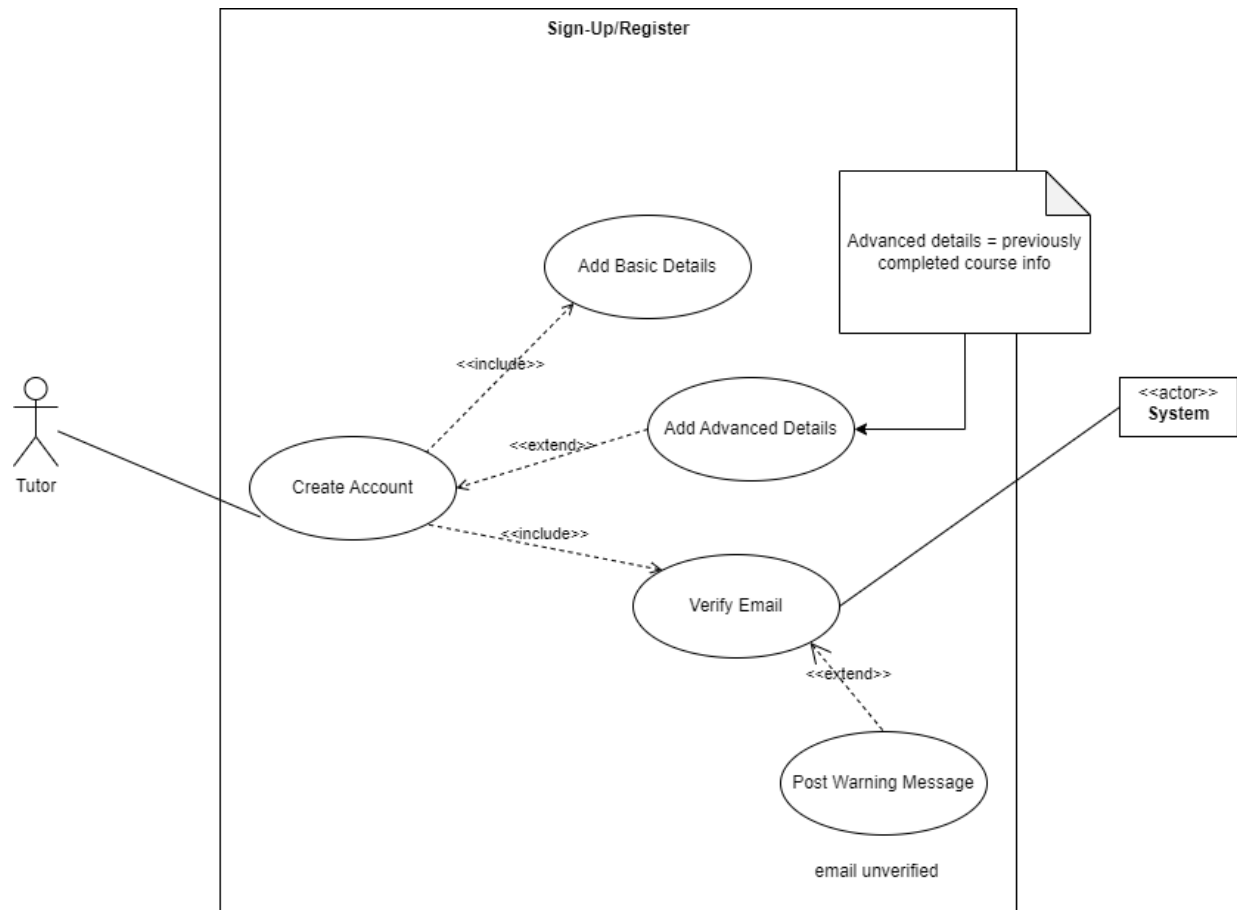


Diagram 3: Sign-Up/Register Use Case Diagram

“Create <Role> Profile” (Administrator Specific)

As an Administrator on MyTutor, I want to be able to create user profiles, so that they can be added to the system. This is important because all roles other than Tutors and TAs are not created via a sign-up process.

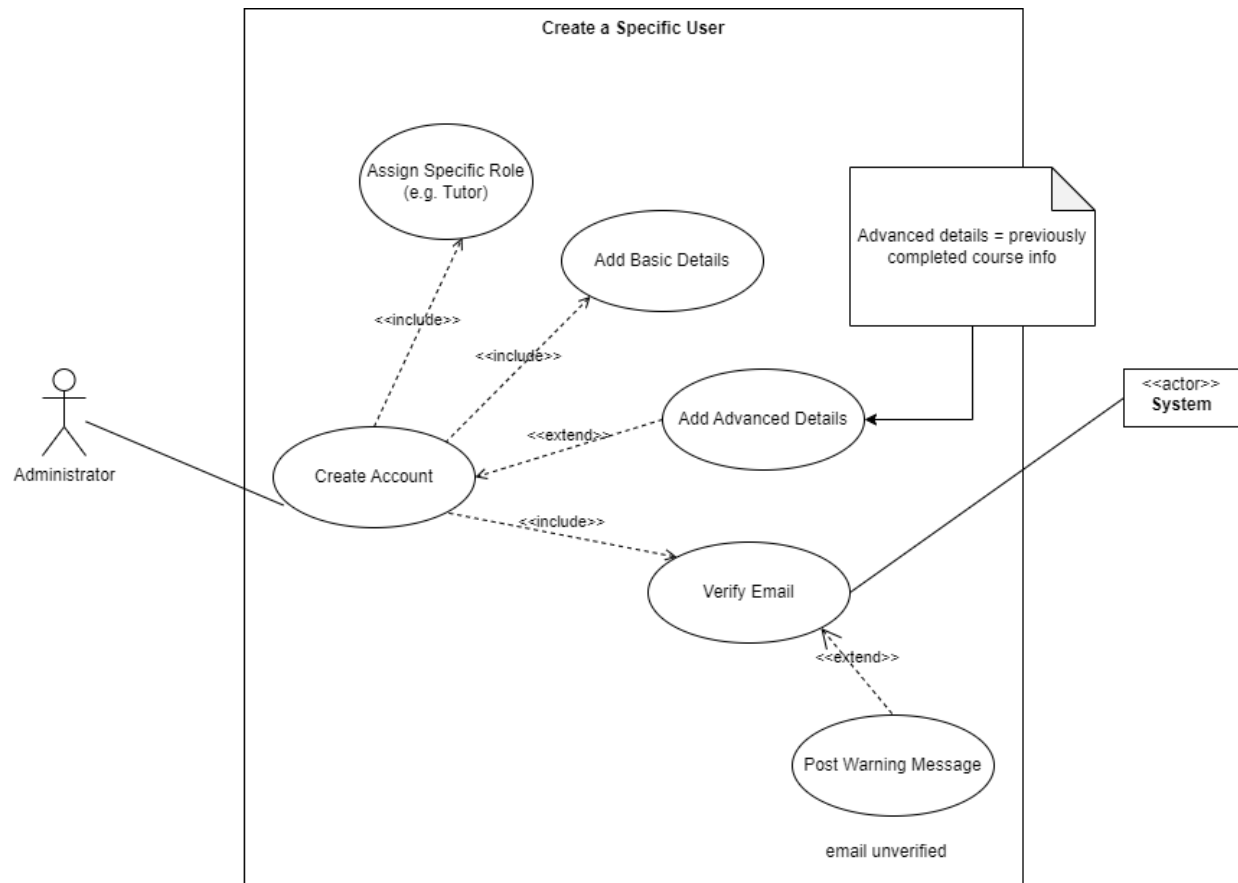


Diagram 4: User Creation Use Case Diagram

“Create and Edit a Course”

As an Administrator in MyTutor, I want to be able to create new Courses, so that associated staff for that Course can be linked and complete actions needed for that Course.

As an Administrator on MyTutor, I want to be able to edit course details, so that I can keep the information accurate and up-to-date.

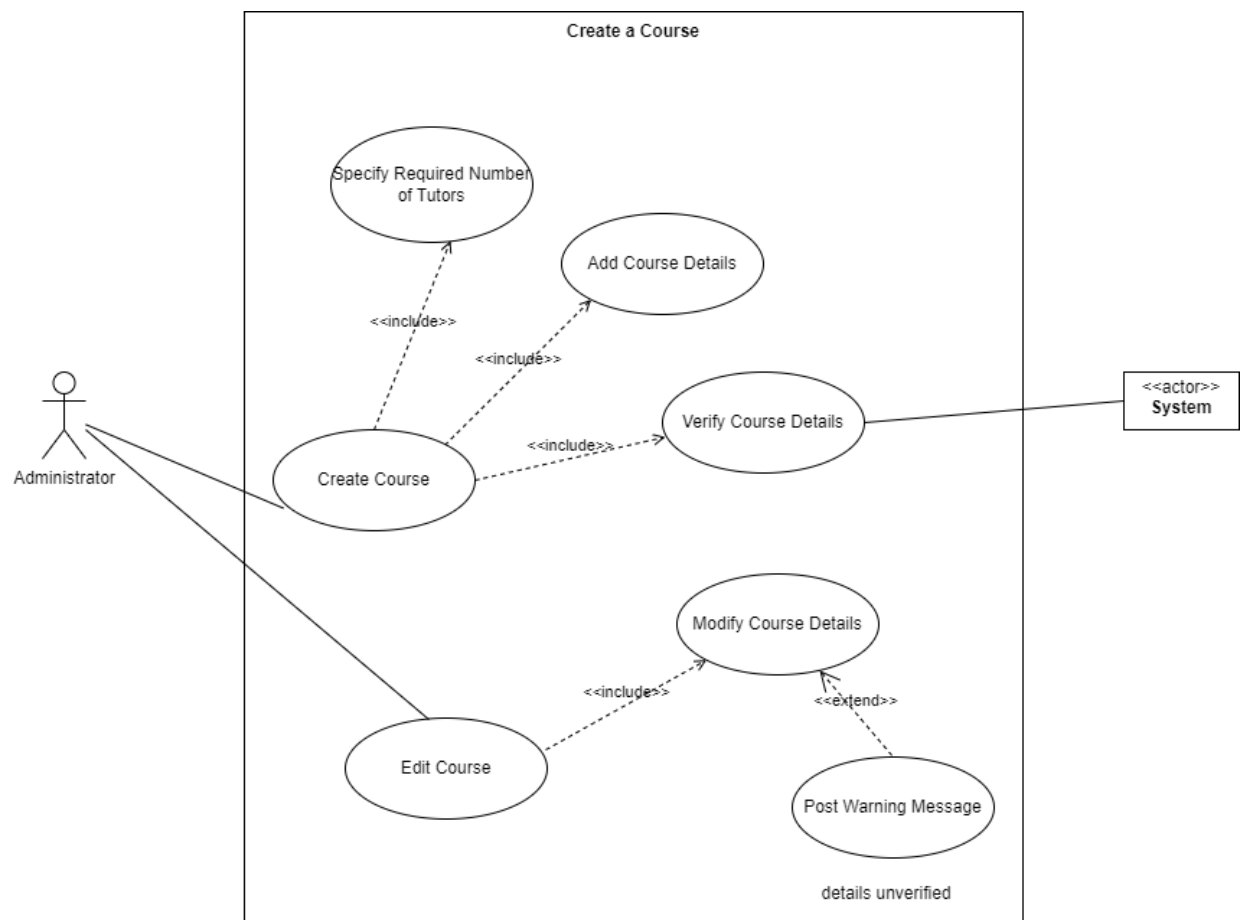


Diagram 5: Course Creation Use Case Diagram

Course Convenor Flow

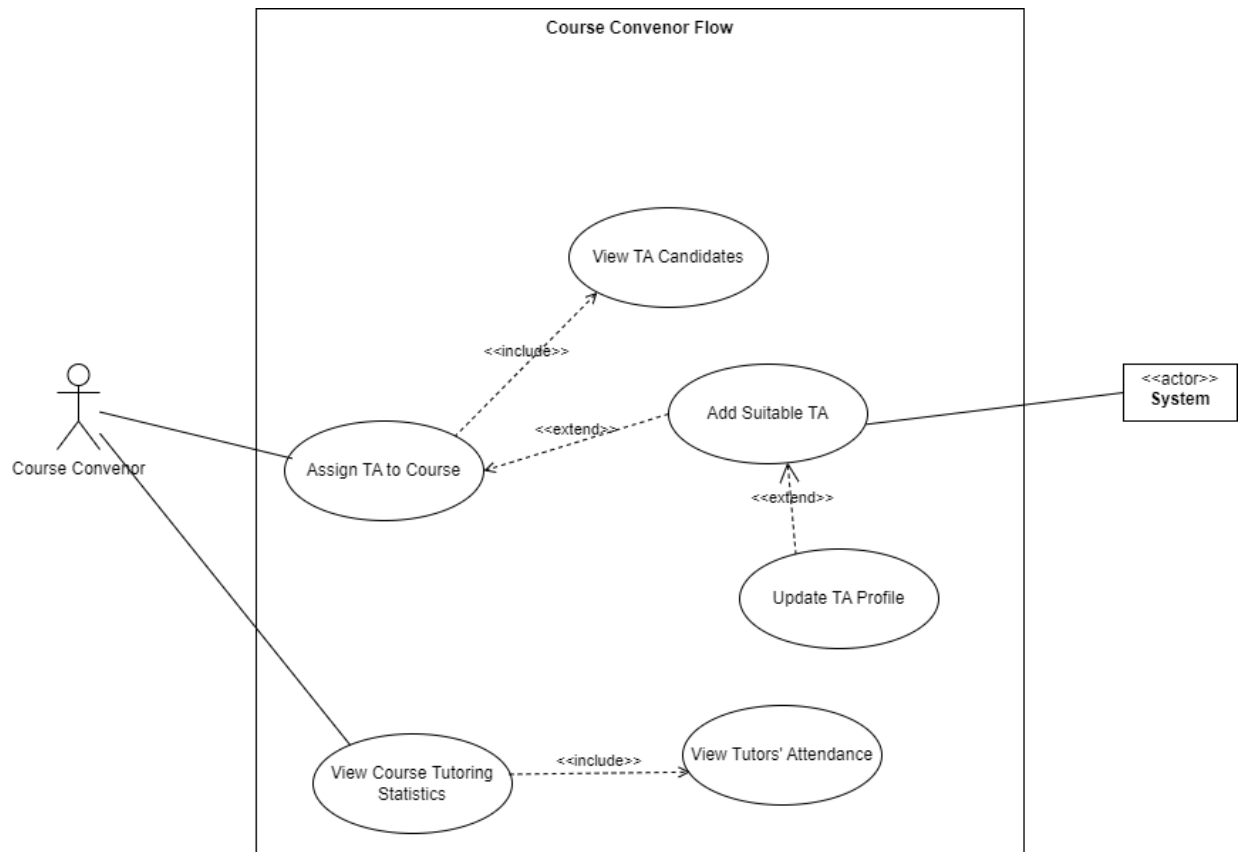
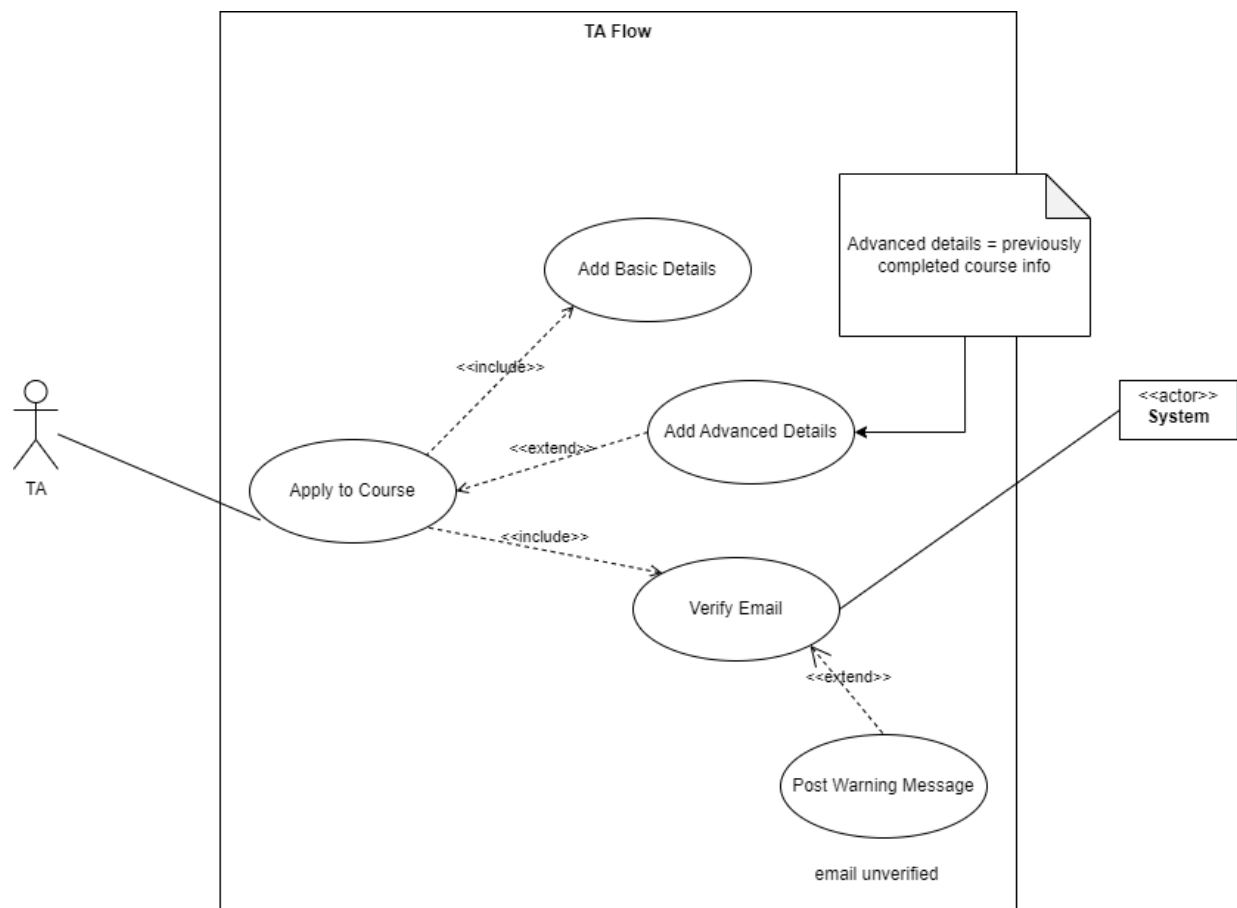
**Diagram 6:** Course Convenor Use Case Diagram**Course Convenor and TA Flow**

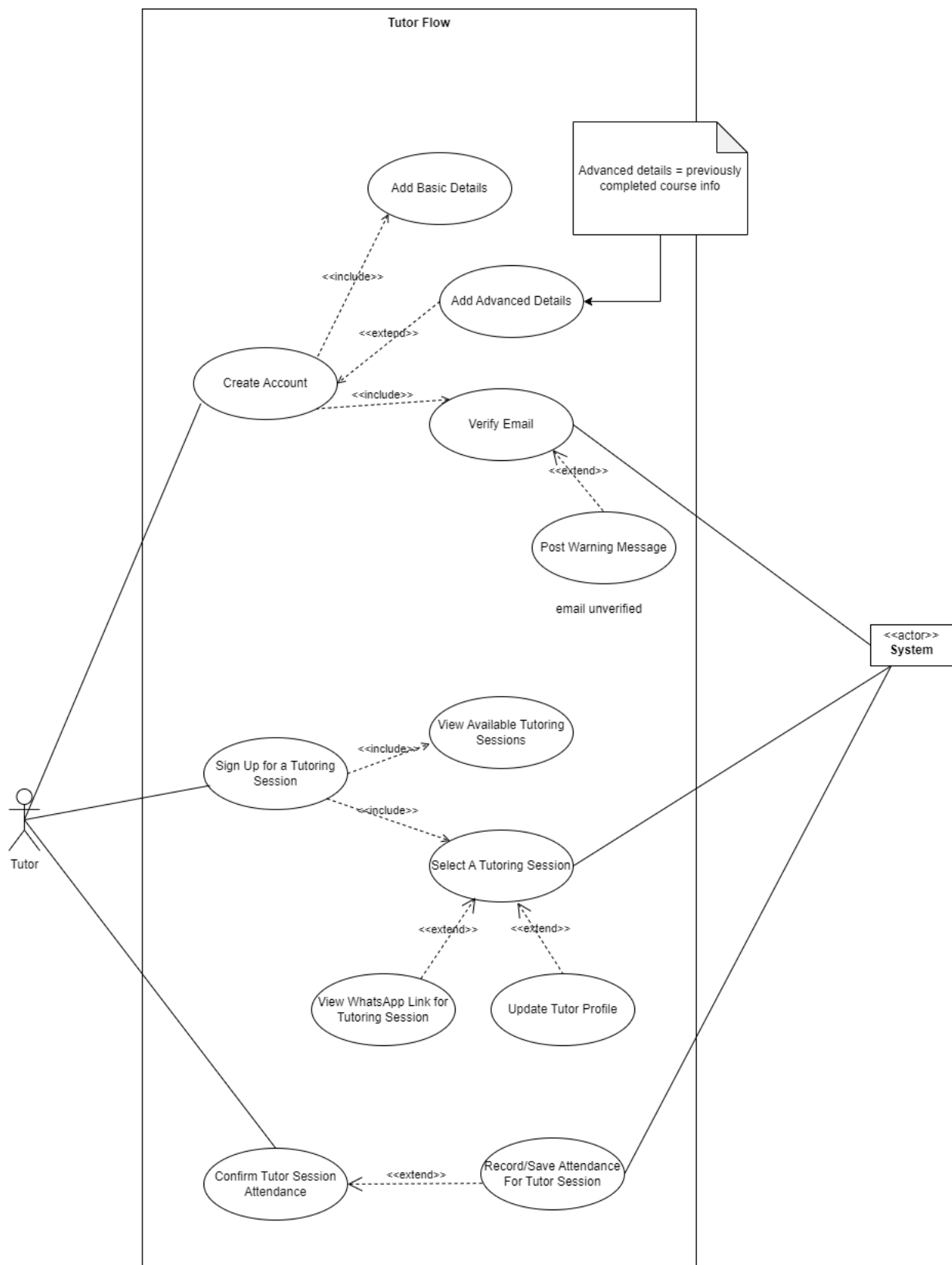


Diagram 7: Course Convenor and TA Use Case Diagram

“Apply to Assist a Course” (TA Specific)

As a TA on MyTutor, I want to be able to apply to assist the teaching of a specific Course, so that I can contribute to the Course's teaching activities.

**Diagram 8: TA Use Case Diagram****Tutor Flow**

**Diagram 9: Tutor Use Case Diagram**

2.2. Description of Major Analysis Artefacts

In this section, we present an overview of the key analysis artefacts for our MyTutor system, which includes Use Case Diagrams and an Entity-Relationship Diagram (ERD).

For the “Sign-In” use case, all users are required to provide their email address and password for authentication. If successful, the system saves the credentials and stores it in a VaadinSession, enabling user access according to roles until they logout.

For the “View a Course and Schedule” use case, all users can view available courses, course schedules, and related course & tutoring information. Users are also just able to view available tutoring sessions for specific courses.

For the Administrator specific use cases, we see that they have the ability to create user accounts, which are role based. Administrators are then able to populate these new user objects, and send it through to be stored in the MyTutor database. They are also able to create new courses to be added to the database. With this, they can specify the relevant data required for the course objects, such as general course details, and number of tutors required. After the creation of a course, the Administrators are also able to edit the course, by modifying the applicable details.

Looking at the “Course Convenor Flow” use case diagram, we see that Course Convenors can assign teaching assistants to their course, as well as viewing their course’s tutoring statistics, such as the tutors’ attendance.

With the “Course Convenor and TA Flow”, we see that both roles can accept tutors into their course, as well as creating a course schedule, by allocating tutoring sessions on certain days, at set times and venues. These course schedules can also be edited by the course convenor or teaching assistant, by updating the dates and times of the tutoring sessions, as well as the venues in some instances

For the TA use case diagram, we see that they also can apply to a course. In this case, they are required to add their required details, as well as previously completed courses to validate their eligibility to become a teaching assistant.

Finally, we have a look at the “Tutor Flow” use case diagram, where we see that students (possible future tutors) can create an account on the MyTutor application, sign up for a tutoring session, and also confirm their attendance at the tutoring session. When creating their account, students provide the relevant information, which will be used to validate their eligibility to tutor a course. When signing up for a tutoring session, tutors are to select an available time slot of their choice, on their preferred day. After this is confirmed and they start tutoring, tutors will then be required to confirm attendance at each tutoring session, every week.

In Section 4.1 of our report, we have our Entity-Relationship Diagram on display, and looking at the diagram, we notice many intricate relationships amongst the database tables. Firstly, we see that one student can have many applications to become a tutor. A requirement to become a tutor would be that the student must have successfully completed the course that they wish to tutor. Thus, one student can have many completed courses, which are listed in their application to validate their eligibility to tutor. If successfully accepted, a student can become a tutor, and a tutor can have a tutoring session.

Looking at the tutoring sessions, knowing that we have multiple tutors, we see that one tutoring session can have one or more attendance records, completed by each tutor present.

For courses, we see that one course can have many tutors, as well as many teaching assistants, and also many tutoring schedules.

Finally, looking at the employees, we see that one employee can have one or more accessible courses, and with administrators specifically, we see that they have no relation with the other database tables. This is due to the fact that they control the processes within the MyTutor system, and also because they generally are not actively involved in the teaching processes of the departments at UCT.

3. Design Overview

Our design for MyTutor was meticulously crafted to seamlessly align with the expected behaviour of the final product increment, whilst simultaneously ensuring the development of a robust, secure, and user-friendly platform. The project team aimed to meet the diverse requirements set forth by our client while placing paramount importance on efficiency, scalability, and security.

The design choices made by our project team was driven by several key considerations:

- **Scalability:** To accommodate the potential growth of users, courses, and tutoring sessions, we adopted a modular and layered architecture. This design not only addresses the current requirement, but also anticipates future enhancements without disrupting existing functionality.
- **Security:** Given the sensitivity of user data, we implemented strict role-based access controls. During the initialization of each view, the system checks the user's authorisation status. Unauthorised users are promptly signed out, redirected to the sign-in page, and presented with an 'Access Denied' dialog, thus ensuring data security and integrity.
- **Usability:** Our relentless focus on a user-friendly interface with intuitive navigation guarantees that users, irrespective of their roles, can effortlessly perform tasks within the system. This emphasis on user experience underpins the system's accessibility and user satisfaction.
- **Maintainability:** The Design Class Diagram reflects our unwavering commitment to object-oriented design principles. By adhering to the SOLID Open/Closed Principle, we designed classes that can be extended rather than modified, enhancing the codebase's maintainability and extensibility.
- **Efficiency:** In an effort to optimise system performance and resource management via hosting on Google Cloud, the use of a layered architecture demonstrates a separation of concerns, which has been proven to provide better performance.
- **Flexibility:** We structured the system around modules of microservices and layers, providing adaptability to changing requirements and accommodating potential future enhancements with ease.

The MyTutor design is justified in terms of the expected behaviour of the final product, as it provides a clear and intuitive interface for users, whilst ensuring data integrity and security. Significantly, the system is designed to handle multiple users and roles, with different levels of access and functionality based on the user's role. This section provides an overview of our design choices and the rationale behind them.

3.1. Design Class Diagram

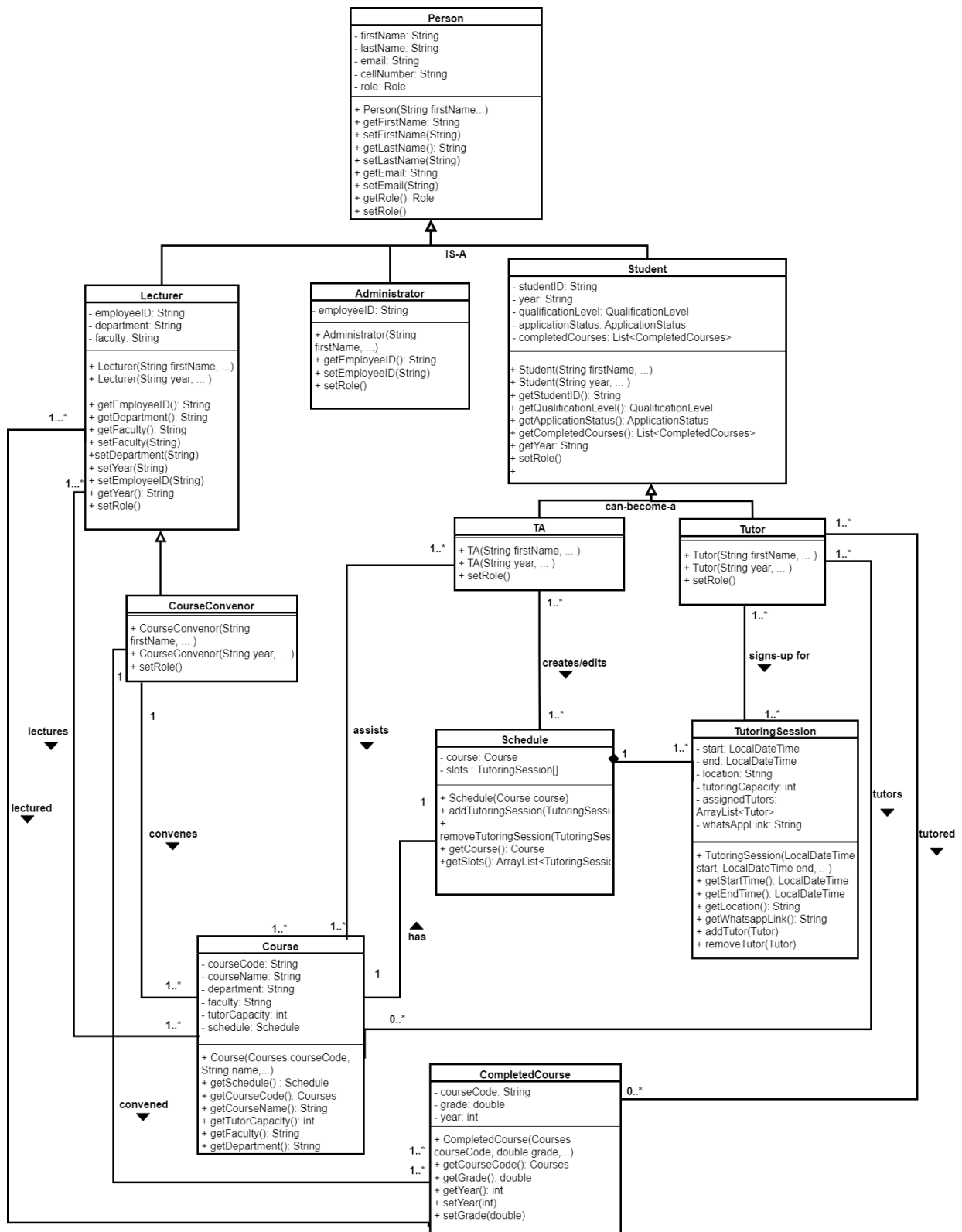


Diagram 10: Design Class Diagram

The Design Class Diagram provides a visual representation of the key classes and their relationships within the MyTutor system. It provides a comprehensive view of the system's architecture, highlighting the essential components and their interactions. Within this diagram, we identify critical entity classes like Administrator, Lecturer, and Student. Additionally, we showcase UI classes such as UserManagementView, microservice classes including ScheduleManager and CourseManager, acting as intermediaries between the front-end and back-end, and the DatabaseController class, responsible for database connectivity. This diagram was instrumental in structuring the codebase and ensuring a modular and maintainable design.

3.2. Layered Architecture Diagram

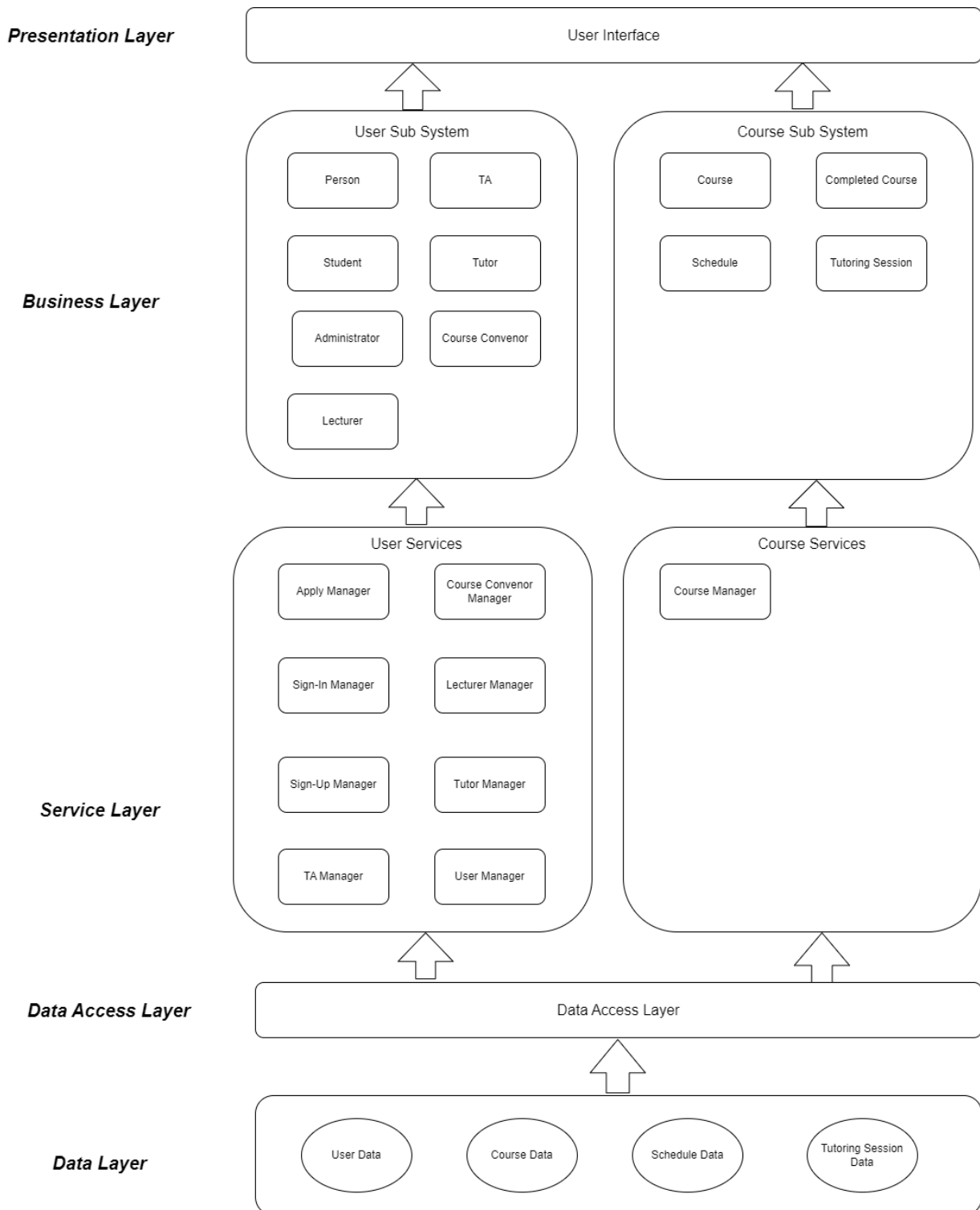


Diagram 11: Architecture Diagram

The Layered Architecture Diagram provides a high-level visualisation of the MyTutor system's architecture, emphasising the separation of concerns. Each layer serves a distinct purpose and interacts harmoniously with adjacent layers, collectively enhancing the system's overall functionality and robustness. This architectural arrangement promotes seamless communication between layers. Notably, the Presentation Layer, encompassing all View classes in our codebase, engages directly with users,

handling data presentation. The Business Layer embodies the core system functionality, housing entity classes and microservices. The Data Access Layer, tightly integrated with our DatabaseController class, handles data storage and retrieval from the database. The layered architecture affords flexibility, enabling modifications in one layer without affecting others, thereby bolstering the system's adaptability and maintainability.

In summary, our MyTutor system design embodies a holistic approach that successfully justifies itself against the expected behaviour of the final product. It prioritises user experience, security, scalability, and efficiency, ensuring a solid foundation for an effective tutoring management platform.

4. Implementation

MyTutor is developed using Java, with a strong focus on adhering to object-oriented design principles. To realise the web-based application, we employed the Vaadin web-framework alongside Spring Boot. The system interacts with a secure MySQL database that is hosted on the UCT Computer Science's Nightmare SSH server, ensuring data persistence, integrity, and confidentiality. Additionally, the web application is hosted on the dependable and scalable Google Cloud infrastructure.

4.1. Data Structures

The data structures used in our systems are primarily represented as classes and enums. Classes are used to represent different entities in our system, such as Person, Student, Lecturer, and CompletedCourse. Enums are used to represent various states and categories in the system, such as Response, ApplicationStatus, QualificationLevel, and Role. In terms of large-scale data structures, a variety of lists and arrays are used as an efficient way to transport data between design layers. In essence, the most important 'data structure' of sorts is the relational database at the centre of the system.

Database and its Organisation & Normalisation

At the heart of MyTutor lies a secure MySQL relational database hosted on the UCT Computer Science Nightmare SSH server. This infrastructure serves as the backbone for data persistence, integrity, and confidentiality. The system seamlessly interacts with this database to securely store and retrieve user data.

To justify our choice of MySQL, one only needs to look at its advantages of robust data integrity, scalability to accommodate more users, data persistence, and versatility. In particular, one of our deciding factors was MySQL's excellence in handling complex data relationships, while simultaneously ensuring data consistency. Additionally, for hosting purposes, MySQL was the best option as we were able to freely host our database on UCT's Nightmare SSH server, and with this, we were able to easily integrate the database and connection to the server into our application.

The database offers tables for the different user roles, courses, schedules, and more as seen in our ERD. This data organisation simplifies the tracking of relationships between data, facilitating easy storage and retrieval.

Key tables in the database include:

- **Students:** Stores student data after initial sign-up on MyTutor, containing fields like studentID, firstName, lastName, email, password, qualificationLevel, and applicationStatus.
- **Employees:** Stores employee data of all employees registered on the application, with fields such as employeeID, firstName, lastName, email, password, department, and faculty.

- **Courses:** Manages course data of all the courses that have tutoring services, including fields such as courseCode, courseName, tutorCapacity, and TACapacity.
- **Administrators:** Stores administrator data, including fields such as employeeID, firstName, lastName, email, and password.
- **Schedules:** Keeps track of tutoring schedules, each associated with a specific course and year, containing fields like scheduleID, courseCode, and year.
- **TutoringSessions:** Stores data for each tutoring session within a specific schedule, with fields such as tutSessionID, scheduleID, day, startTime, endTime, tutoringCapacity, location, and whatsappLink.
- **AccessibleCourses:** Stores data of employees and the course that they are either lecturing or convening, as well as the year in which the role was performed, containing fields such as empID (foreign key referencing the Employees table), courseCode (foreign key referencing the Courses table), role, and year.
- **CompletedCourses:** Stores data of students and the relevant courses that they have completed, as well as their grade and the year in which they completed the course, containing fields such as courseCode (foreign key referencing the Courses table), studentID (foreign key referencing the Students table), grade, and year.
- **TutoringSessionTutors:** Stores the data of tutors and the course that they are tutoring, containing fields such as tutSessionID (foreign key referencing the TutoringSessions table), and studentID (foreign key referencing the Students table).

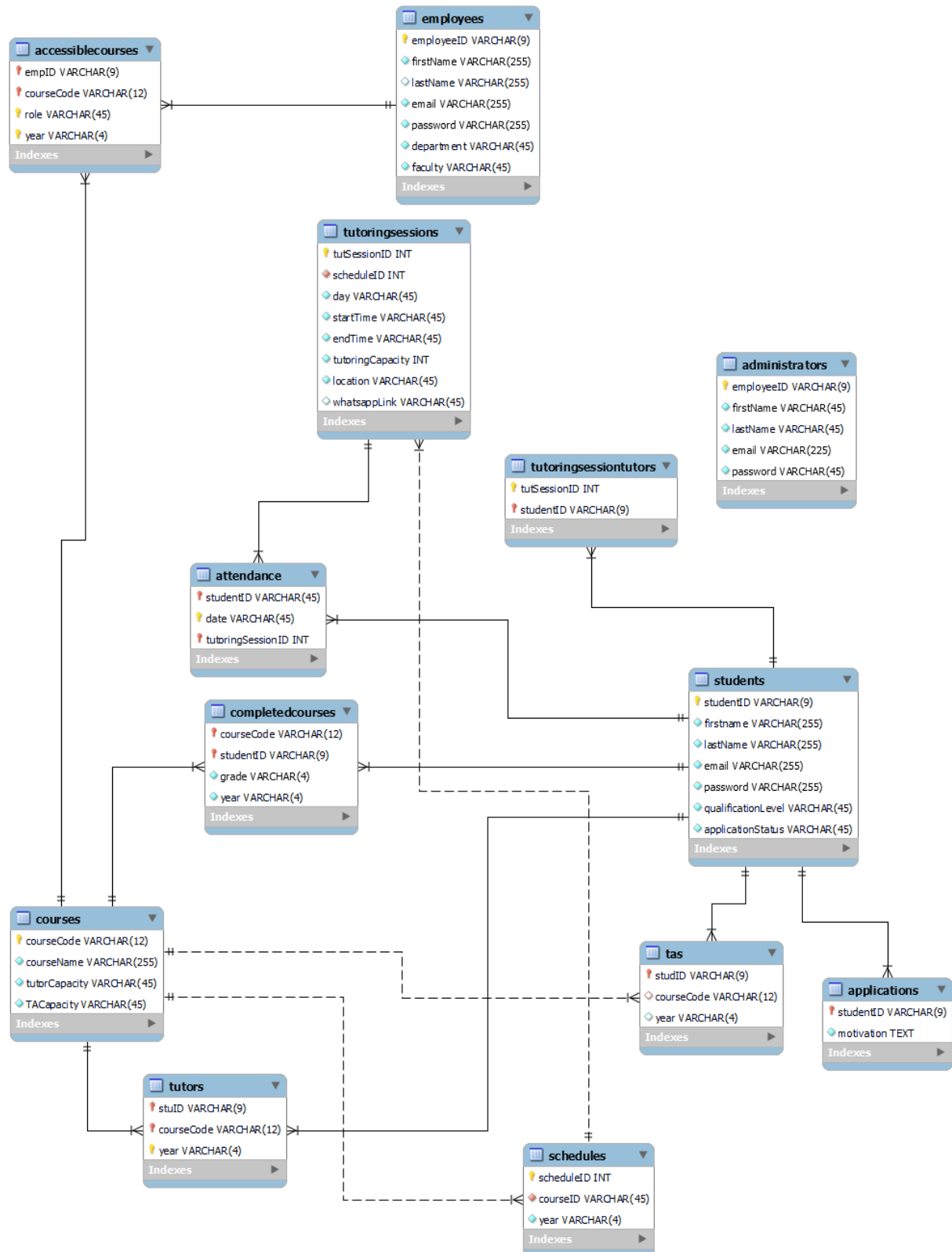


Diagram 12: ERD Diagram

Additionally, the database includes tables with composite keys to manage data uniquely identified by multiple columns across various tables, including Tutors, TAs, AccessibleCourses, Applications, CompletedCourses, Attendance, TutoringSessionTutors.

As is good database design practice, we have applied 3rd-Normal-Form to our database to eliminate data redundancy, improve data integrity, and minimise the risk of data inconsistencies. This approach enhances data consistency and maintains the quality of the data in our database.

4.2. User Interface

When it comes to the scope of the MyTutor project, the need for an intuitive User Interface arises naturally. In order to optimise and tailor the experience of a user, be it a student, tutor, TA, Lecturer or Course Convenor, we needed to implement a User Interface to further streamline the transactions each of these roles would have with the system.

The user interfaces for MyTutor were constructed using Vaadin, a Java framework tailored for developing web-applications. The UI design is responsive, ensuring compatibility with various screen sizes. Prioritising simplicity and usability, the interfaces offer easy navigation for all users. A consistent colour scheme and typography reinforce and emphasise the system's unique identity. All of this is aided by the using Vaadin's layout components, like FlexLayout, FormLayout, HorizontalLayout, and VerticalLayout. One of the most advantageous aspects of using Vaadin is that it allows the UI to be written completely in Java, which greatly simplifies the development process.

From a user experience perspective, MyTutor ensures smooth navigation within the system, allowing users to efficiently perform their roles and responsibilities. For example, Tutors can easily select tutoring sessions that align with their availability through a user-friendly UI. The system incorporates feedback mechanisms through dialogs to inform users of the outcomes of their actions, or if any errors occurred.

With Vaadin, the system also has strong authentication and authorisation, requiring users to sign in with their email addresses and passwords. With this, secure VaadinSessions are implemented to prevent unauthorised access, allowing each user and their role to be recognised for the duration of their session.

4.3. Significant Procedures

MyTutor is divided into several classes, each with its own set of methods that perform specific tasks. In the interest of discussing the most significant methods in each class, we focus on key classes and their essential methods:

ENTITY CLASSES

Person

- **getFirstName():** Retrieves the first name of a user object.
- **getLastName():** Retrieves the last name of a user object.
- **getRole():** Retrieves the role of a user object.

Administrator

- **Administrator():** Instantiates an Administrator object whenever a new administrator is to be used or be added to/fetched from the Administrators table in the database.
- **getEmployeeID():** Retrieves an employee's unique ID.

Lecturer (and Course Convenor)

- **Lecturer():** Instantiates a Lecturer object whenever a new lecturer is to be used or added to/fetched from the Lecturers table in the database; casted to Course Convenor when necessary.
- **getEmployeeID():** Retrieves an employee's unique ID.

Student (Tutor and TA)

- **Student():** Instantiates a Student object whenever a new student is to be used or added to/fetched from the Students table in the database; casted to Tutor or TA when necessary.
- **getStudentID():** Retrieves a student's unique ID.
- **getQualificationLevel():** Retrieves the student's qualification level, like ThirdYear, Masters, etc.
- **getCompletedCourses():** Retrieves a list of a student's previously completed courses, which determines whether they are eligible to tutor a specific course:

Course

- **Course():** Instantiates a Course object whenever a new course is to be used or added to/fetched from the Course table in the database.
- **getSchedule():** Retrieves the schedule for the specific course, which was created and edited by the Course Convenor or TA.
- **getCourseCode():** Retrieves the course code, the unique course code to identify the course which is being referenced.
- **getName():** Retrieves the course name.

- **getTutorCapacity():** Retrieves the tutor capacity, the maximum amount of tutors to be accepted to tutor for the course.
- **getTaCapacity():** Retrieves the TA capacity, the maximum amount of teaching assistants to be accepted to assist for the course.

Schedule

- **Schedule():** Instantiates a Schedule object whenever a new Schedule is to be used or added to/fetched from the Schedules table in the database.
- **addTutoringSession():** Adds a TutoringSession, with a specific time slot, to an existing Schedule.
- **removeTutoringSession():** Removes a TutoringSession, with a specific time slot, from an existing Schedule.
- **getScheduleID():** Retrieves the unique schedule ID.
- **getTutoringSessions():** Retrieves a list of the TutoringSessions for the specific Schedule.
- **getCourse():** Retrieves the Course for which the Schedule is associated with.

CompletedCourse

- **CompletedCourse():** Instantiates a CompletedCourse object whenever a new CompletedCourse is to be used or added to/fetched from the CompletedCourses table in the database.
- **getCourseCode():** Retrieves the course code of the Course that the Student completed.
- **setCourseCode():** Sets the course code of the Course that the Student completed.
- **getGrade():** Retrieves the grade that the Student achieved for the Course.
- **setGrade():** Sets the grade that the Student achieved for the Course.
- **getYear():** Retrieves the year when the Student completed this Course.
- **setYear():** Sets the year when the Student completed this Course.

TutoringSession

- **TutoringSession():** Instantiates a TutoringSession object whenever a new TutoringSession is to be used or added to/fetched from the TutoringSessions table in the database.
- **getSessionID():** Retrieves the session ID.
- **getStartTimeAsDouble():** Retrieves the start time as a double.
- **getEndTimeAsDouble():** Retrieves the end time as a double.
- **getStartTimeAsLocalTime():** Retrieves the start time as a LocalTime object.
- **getEndTimeAsLocalTime():** Retrieves the end time as a LocalTime object.
- **getDay():** Retrieves the day.
- **getWhatsappLink():** Retrieves the WhatsApp link.

- **getTutoringCapacity():** Retrieves the tutoring capacity.
- **getSignedUpTutors():** Retrieves the list of tutors that are signed-up for the TutoringSession.
- **addTutor():** Adds a tutor to the list of signed-up tutors.
- **removeTutor():** Removes a tutor from the list of signed-up tutors.
- **getLocation():** Retrieves the location.

UI CLASSES

A significant method that is used in all UI classes is the `beforeEnter` method. It is used for security authentication in such a way that before the view is initialised, the `VaadinSession` is checked to ensure that the current user has been authenticated and signed-in, or that they are of the correct role, i.e. authorised, to be viewing a specific view.

WelcomeView

The `WelcomeView` class is the entry point of the MyTutor application, providing a user interface for users to sign in or sign up. It includes security authentication, input fields for email and password, and buttons for sign-in and sign-up, with corresponding actions such as validation, interaction with the database, and navigation to other views based on the user's actions.

ProfileView

The `ProfileView` class primarily serves to display and manage user profiles within MyTutor. It provides functionality to retrieve completed courses for students, create an Edit button with interactive editing capabilities, and create a temporary `Person` object to hold profile details before modifications. These procedures enhance the user experience by allowing profile management and editing within the application.

- **getCompletedCourses():** If the current user is a Student, this method is used to retrieve the list of `CompletedCourses` associated with the user from the `VaadinSession`, to be displayed in the `CompletedCourses` Grid component.
- **createEditButton():** Creates and configures an Edit button with an associated `ClickListener` that toggles between enabling `TextFields` for editing, and saving the changes made in the `TextFields`.
- **editClick():** Updates the current user's profile and displays a success or error message.
- **createNewPerson():** Creates a new `Person` object to temporarily hold the current user's profile details before changes have been made.

ApplicationView

The `ApplicationView` class represents the application view of the MyTutor application, where students who are signed up can apply to be considered for upcoming Tutor/TA acceptances. It provides functionality for students to submit their application with a motivation statement, and handles different application states such as when a student has not yet applied or has already applied.

- **applyClick():** Retrieves the current student from the `VaadinSession` and uses it to submit an application with a given motivation.

UserManagementView

UserManagementView is a class that provides a user interface for managing users in the application. It allows administrators to create, edit, and delete user profiles for other administrators, course convenors, and lecturers, and also provides functionality for filtering users based on their user ID.

CourseManagementView

The CourseManagementView class is a part of the MyTutor UI that allows Administrators to manage courses in the MyTutor application. It provides functionalities such as creating, editing, and deleting courses, as well as adding employees to courses, all through a user-friendly graphical interface.

- **createClick():** Creates a new course object.
- **editClick():** Updates a selected course object with new course information

CoursesView

The CoursesView class is responsible for creating a view that displays a list of active courses as clickable blocks, with each block navigating to the corresponding CourseView when clicked. It includes special procedures for initialising the list of courses based on the user's role, creating a search field for course codes, filtering courses based on a given course code, and updating the course blocks accordingly.

CourseView

The CourseView class represents the view for a specific course in a tutoring application, displaying various sections related to the course such as schedule, tutor listing, TA listing, and course information. It includes special procedures for handling user interactions such as tab selection, schedule editing, and displaying dialogs for tutoring statistics, course applications, and student motivations.

MainLayout

The MainLayout class defines the main layout of the application, which acts as a top-level placeholder for other views. It includes special procedures for initialising user details from the session, adding header and drawer content, toggling between dark and light themes, handling user sign-out, creating navigation menus, and updating the view title after navigation.

SignUpView

The SignUpView class is responsible for creating and managing the sign-up form in the MyTutor application. It provides fields for user details and completed courses, handles form submission, and interacts with the SignUpManager and SignInManager microservices to register and authenticate users.

MICROSERVICE CLASSES

UserManager

The general purpose of the UserManager class is to deal with entity classes relating to users by serving as a communication medium between the frontend administrator view and the database. It implements various methods mapping out the CRUD functionality the database possesses as well. This is done in light of the separation of concerns design principle.

- **get<X>():** These methods return a list of a particular type (eg: getAdministrators() returns a list of administrators). This is generally true for all the entity classes as needed by the frontend to display the appropriate data to the user.
- **create():** Creates a new user (role specific), and adds the user into the database.

- **generatePassword():** Generates a random password for each new user being added to the system.
- **validateUserCreation():** Validates the user based on its specific role type. It does this by passing the relevant data through a series of “filters” in order to catch errors.

EmployeeManager

Much like the UserManager, the EmployeeManager groups similar functionality relating to the manipulation of Employee entities in the system. It also provides the relevant communication with the database to ensure synchronous and accurate data transportation from the database to the frontend.

- **doesEmployeeExist():** Records which Employees are added to a specific course for a specific year and returns the appropriate response.
- **addEmpToCourse():** Add Employees to a specific course, and populate the database with the relevant data thereafter.
- **getEmployeesFor():** Retrieve Employees for a specific course.

StudentManager

The StudentManager class provides functionality for managing student-related data and interactions within the system. It includes special procedures such as retrieving attendance statistics for a specific student in a particular course, resetting the system for a new academic year, obtaining lists of active Tutors and TAs for specific courses and years, retrieving a list of pending applicants who have completed a given course, and fetching a student's grade for a specific course.

- **getTutorStatsFor():** Returns statistics in the form of attendance count for a particular course, for a particular student.
- **resetSystem():** Resets a system such that it can be used for a new year.
- **getActiveTutors():** Returns a list of active Tutors for a specific course, for a specific year.
- **getActiveTAs():** Returns a list of active TAs for a specific course, for a specific year.
- **getPendingApplicants():** Returns a list of all applicants who have completed a given course.
- **getGradeFor():** Returns the grade that a student has achieved for a certain course.

SignInManager

This class groups the functionality relating to the sign in of users. It makes sure that all data that is retrieved is secure and that sign in can happen on a role-specific basis.

- **signIn():** Validates and performs a user signing in.
- **validateEmailFormat():** Validates the user's sign-in details
- **setSession():** Sets the session attributes based on the user's role, storing them in the VaadinSession.

SignUpManager

The SignUpManager is the driver class behind the process of adding a new student to the system. When a student wants to sign up to become a tutor / TA, they do so by interacting (indirectly) with this class.

Here, we group together all the functionality needed to manipulate the persistent data storage to reflect the act of adding a new user to the system.

- **signUp():** Allows a new user (Student) to sign up to MyTutor, for the opportunity to apply to become a tutor.
- **validateSignUp():** Validates information provided by the new user.

ApplyManager

The ApplyManager allows a user to make an application to become a tutor / ta. It does this by first validating the application and then performing the actual application manipulation. The action of accepting an application is also defined here.

- **apply():** Allows students to apply to become tutors, based on whether they qualify to tutor a specified course.
- **validateApplication():** Validates an application made by a student based on their qualification level.
- **acceptStudents():** Accepts a student for a specified course, allowing them to tutor the course for the current year.

CourseManager

CourseManager is the grouping of all the functionality related to the manipulation of courses. This, too, is a reflection of the Database's CRUD functionality surrounding a course entity.

- **getCourseCodes():** Returns a list of course codes.
- **getCourses():** Returns a list of all courses as listed in the database.
- **getCoursesFor():** Returns a list of courses for a given employee or student.
- **create():** Creates a new course on the system and updates the database as needed.
- **getCourseStatsFor():** Returns the statistics for a given course.

ScheduleManager

The ScheduleManager class is the centre point of a great deal of the system's functionality. Using this microservice class, a user is able to interact with and manipulate a schedule object. The manipulation is then captured in persistent data for later retrieval.

- **getSchedule():** Returns a Schedule object as stored in the database.
- **tutorSignUp():** Allows a tutor to sign up for a specific tutoring session.
- **tutorCheckIn():** Allows a tutor to confirm their attendance to a specific tutoring session.
- **hasCheckedIn():** Conveys whether a tutor has signed in to a particular tutoring session or not.
- **getTutoringSessionStatsFor():** Obtains statistics (attendance count) for a specific tutoring session

DatabaseController

The DatabaseController truly is the heart of the MyTutor application. It contains all the methods representing the CRUD functionality across all of the entity classes. It also contains further manipulations and thereby fully reflects the capabilities of the system in general. It houses standard components such as setting up the connection to a hosted MySQL database, as well as specific methods defined to obtain, change and remove data as and when the rest of the system needs it.

- **getConnection():** Returns a connection to the MyTutor database.
- **resetSystem():** Resets the system to allow students to re-apply to become tutors during the following year.
- **studentSignUp():** Adds new Student into the database.
- **getStudent():** Retrieves a Student from the database.
- **getEmployee():** Retrieves an Employee from the database.
- **getAdministrator():** Retrieves an Administrator from the database.
- **getAllLecturers():** Retrieves a list of all Lecturers from the database.
- **getAllCourseConvenors():** Retrieves a list of all Course Convenors from the database.
- **getAllTutors():** Retrieves a list of all Tutors from the database.
- **getAllTAs():** Retrieves a list of all TAs from the database.
- **createUser():** Inserts a new user into the database based on their role type.
- **getAllCourses():** Retrieves a list of all courses from the database.
- **createCourse():** Creates a new course and adds it to the Courses table in the database.
- **createSchedule():** Create a course schedule and add it to the relevant database tables.
- **getAssignedTutors():** Retrieve the tutors assigned to the specific tutoring session.
- **getTutoringSessions():** Retrieve the relevant tutoring sessions for the specified schedule.
- **addTutorToSession():** Add a tutor to a specified tutoring session.
- **getTutorStatsFor():** Retrieve the recorded statistics of a tutor for a specific course.
- **getTutoringSessionStatsFor():** Retrieve the tutoring session statistics that have been recorded for that tutoring session.

4.4. Class Relationships

As required by our client, MyTutor was to be designed using object-oriented object principles. Thus, the most prominent relationship between classes within the system is inheritance. The classes in our system have relationships based on the real-world entities they represent. For each user role in the system, there exists a Person class serving as the base class, which sets up the basic structure of the various users of the system. These users range from Administrators, through to Tutors, which all act as subclasses. With regard to the subclasses, the Lecturer and Student classes each also act as a base class to the Course Convenor class, and the Tutor and TA classes respectively. In other words, a Person can be a Student or a Lecturer.

With composition, for example, a Student can have multiple CompletedCourse instances.

Delving deeper into the special relationships between the classes within MyTutor, we have a look at how the database class, DatabaseController, interacts with the microservice classes, such as ApplyManager and SignInManager, which ultimately act as the link between the front-end and back-end of the system.

Specifically, in terms of providing the general user Administrator functionality, the UserManager class interacts with the DatabaseController class to perform database operations to do with the general Person entities.

Looking into the SignInManager, this class interacts with the DatabaseController class to perform database operations that allow every registered user to sign into the system. A VaadinSession is also triggered in this instance, where based on the user role, its respective dashboards are presented within the system. The user's current login credentials are also stored live until they log out of the system.

For the SignUpManager, the class interacts with the DatabaseController class to perform database operations that allow students the opportunity to register to become tutors. After signing up, they can monitor their application status until further information is communicated to them.

4.5. Special Programming Techniques & Libraries Used

Singleton

We made use of the Singleton Design Pattern in our microservice managers, such as UserManager and CourseManager, and our DatabaseController to ensure that, correctly, only one instance of each class is created.

Java Mail API

MyTutor utilises the Java Mail API to send email notifications to users. Specifically, emails containing randomly-generated passwords are sent to users whose accounts have been manually created by an Administrator, such as other Administrators, Course Convenors, Lecturers, and in emergencies, Students. This feature is instrumental in ensuring secure access and communication within the system.

Vaadin, Spring Boot, and Maven with Java

MyTutor was mainly implemented using both Vaadin and Spring Boot, which are tools used to support the development of Java-based web applications which follow Object-Oriented Principles. Maven, a build automation tool, was also utilised in the overall development of MyTutor. This stack of technologies proved to be effective as they each complemented the other, and all offer extensive support for Java projects. With Vaadin, we were able to design and create modern, high quality user interfaces for the MyTutor system with ease. Spring Boot was then implemented to aid in the development of the backend of MyTutor, where the business logic of the system is handled, as well as the database interactions. With Maven, we were able to tie everything together, as the project management tool was used to manage our various dependencies and project builds, thus allowing for the smooth running of our entire system.

5. Program Validation and Verification

The MyTutor project employed a testing system that took place throughout the development process. This materialised as a testing procedure that took place at the end of each development cycle - testing the new and previously implemented functionality. By approaching testing this way, we were able to produce fully functional constituent parts which assemble seamlessly to form the larger system.

We chose to implement the testing procedure this way to ensure that no progress is left untested. It also ensured that testing wasn't left to the end of the project - thereby allowing enough time to adequately debug and fix errors.

Table 1: MyTutor Testing Plan

Process	Technique
1. Unit Testing: test the methods of classes in isolation	Unit Tests
2. Class Testing: test methods and state behaviour of classes	Random, Behavioral and White-Box Tests
3. Integration Testing: test the interaction of sets of classes	Random, Behavioral, and White-Box Tests
4. Validation Testing: test whether client requirements are satisfied	Random, Behavioral, and White-Box Tests
5. Regression Testing: re-run previous tests to ensure code changes do not cause new errors	Random, Behavioral, and White-Box Tests
6. Security Testing: test for weaknesses within the system	Penetration and Behavioral Tests
7. System Testing: test the behaviour of the system as part of a larger environment	Behavioural, Security, White-Box Tests

The testing protocol mentioned above proved to be effective for the MyTutor system, as the system was tested thoroughly using the various testing techniques.

We created unit tests to test some of the core functionality that is unrelated to the Database connection. These tests are in the minority as the majority of the project rests on its interaction with the database. We then carried out integration tests for the various class methods that interacted with each other, mainly by performing random, white-box tests, including behavioural tests to ensure correctness of the system. For validation and security testing, we evaluated the system's robustness against incorrect input data, and more specifically for security testing, against SQL injection attacks. We combatted those attacks by using prepared statements for our SQL queries, and with the normal input data, we developed checking methods to ensure the correctness of the input data. For the regression tests, we mainly conducted the complete, and partial regression tests, to ensure that changes to the code would not negatively impact the sections of the system that are unrelated. Finally, for our system tests, we conducted more random, behavioural, and white-box tests to ensure that the entire system works as required, and that it meets the client's expectations.

Table 2: Tests Carried Out for MyTutor

Data Set and reason for its choice	Test Cases		
	<i>Normal Functioning</i>	<i>Extreme boundary cases</i>	<i>Invalid Data (program should not crash)</i>
Successful User Sign-In	Passed	n/a	Produces Error Message; Prompts for correct data

Unsuccessful User Sign-In	Passed	n/a	Produces Error Message; Prompts for correct data
Course Viewing	Passed	n/a	n/a
Schedule Viewing	Passed	n/a	n/a
<Role> Profile Creation [Administrator Specific]	Passed	n/a	Produces Error Message, Prompts for correct data format
Course Creation [Administrator Specific]	Passed	n/a	Produces Error Message, Prompts for correct data format
Course Editing [Administrator Specific]	Passed	n/a	Produces Error Message, Prompts for correct data format
TA Assignment [Course Convenor Specific]	Passed	n/a	n/a
Tutoring Statistics Viewing [Course Convenor Specific]	Passed	n/a	n/a
Schedule Creation [TA and Course Convenor Specific]	Passed	n/a	Produces Error Message, Prompts for correct data format
Tutor Acceptance [TA and Course Convenor Specific]	Passed	n/a	n/a
Tutor List Editing [TA and Course Convenor Specific]	Passed	n/a	n/a
Adding a WhatsApp Link [TA and Course Convenor Specific]	Passed	n/a	n/a
Tutor Application [Tutor Specific]	Passed	n/a	Produces Error Message, Prompts for correct data format
Tutoring Session Sign-Up [Tutor Specific]	Passed	n/a	n/a
WhatsApp Link Viewing [Tutor Specific]	Passed	n/a	n/a
Attendance Confirmation [Tutor Specific]	Passed	n/a	Produces Error Message, Prompts for correct data format

Following the tests which were conducted above, we were able to assess the extent to how useful, and usable the MyTutor system is. We have run through tests which range from the applications of new tutors, through to the creation of users and courses by the administrator, as well as the creation of course

tutoring schedules by the course convenors/TAs. With the full coverage of tests that were conducted, and the level of debugging and validation done to ensure that no incorrect data can crash the system, we can safely state that the system is fully useful and usable for its intended purpose.

Due to the use of our extensive and detailed testing plan, the completed tests allow us to ensure that the MyTutor system functions as expected and that it handles all user-input gracefully.

6. Conclusion

*Your report must have a clear conclusion where you **revisit the aims** set out in the beginning and discuss **how well you met them**. Did you achieve the objective of creating a well-structured, modular, and robust system? Please summarize the design features and test results that show thi*

The main goals established for MyTutor was to develop a reliable, modular web application that would automate and improve the tutor management process. These goals have been achieved through the development and deployment of the system, and with the help of significant design elements and test data, we analyse how well these goals were accomplished.

The user-friendly design of the MyTutor system allows for easy navigation of the platform for all users, based on their role. Scalability and flexibility are also made possible using a modular design for the system. Throughout the development process, the modular approach has made it simple to customise and expand the system's capability.

In order to determine the robustness of MyTutor, thorough testing was done, regarding the system's reliability. The system contains many validation methods to prevent incorrect data being entered, and data security methods such as access control strengthen the protection of sensitive data within the MyTutor system.

The test results for MyTutor confirmed the complete operation and reliability of the system. During unit testing, components within the system functioned successfully in isolation, indicating correctness within the system. Whenever code changes were made, regression testing was done by re-running previous tests to ensure successful operation of existing components. Security tests allowed for the testing of weaknesses within the system, especially on areas concerning the database, to prevent SQL Injection attacks. Validation tests were also performed, ensuring that client requirements were successfully met. Finally, system tests were conducted to evaluate the overall behaviour of the system, which proved to successfully function as intended.

Throughout the system's development, great attention to detail was paid to both the design and implementation of various artefacts. The result is a system that is reliable, dependable and appropriate according to its defined requirements.

Thus, we can confidently say that the resultant system successfully meets the requirements that were set out at project start by the client: MyTutor provides a comprehensive Tutor Management System that is well-structured, modular, and robust - providing an efficient, simple way to manage tutors in the educational landscape of UCT.

7. References

Logo.com, L.M.B.B. (2023) Logo Maker & Brand Builder.
https://app.logo.com/editor/templates/?editing_logo=logo_0bf5f3e4-1a5f-45b9-a3a0-322213caaceb&logo=logo_0bf5f3e4-1a5f-45b9-a3a0-322213caaceb (24 Aug. 2023)

IBM (no date) What is Java Spring Boot?
<https://www.ibm.com/topics/java-spring-boot#:~:text=Java%20Spring%20Boot%20> (24 Aug. 2023)

Vaadin.com (no date) Overview | Introduction | Framework | Vaadin 8 Docs.
<https://vaadin.com/docs/v8/framework/introduction/intro-overview#:~:text=Vaadin%20Framework%20is%20a%20Java,is%20the%20more%20powerful%20one.> (24 Aug 2023)

Gaba, I. (2023) 'What is Maven: Here's What You Need to Know'.
<https://www.simplilearn.com/tutorials/maven-tutorial/what-is-maven.> (24 Aug 2023)

Appendix A: User Manual

This manual provides a comprehensive guide on how to use the MyTutor software solution. The guide has been partitioned into sections based on the current user's role and status.

1. Student without an Account

You have just received the link to the MyTutor web-application due to your adequate marks for your courses, likely via an email from your department/faculty's administrator. As a student without an account, your first step is to sign-up.

Sign Up

1. On the landing page, click the "Sign Up" button.
2. Fill in all required details on the form, such as your student ID, email, and add all the courses that you have completed at UCT, with the associated year and grade you received.
3. Click the "Submit" button.
4. Upon successful registration, you will see a confirmation dialog, and you will be automatically signed in.
5. Congratulations, you now have a MyTutor account!

2. Student with an Account and No Active Application

You have just successfully signed-up and have a MyTutor account. Now, in order to be included in the pool of applicants to be considered to be accepted as a Tutor or TA for a course that you have completed, you must submit an application.

Submit an Application

1. On the landing page, fill in your email and password.
2. Click on the "Sign In" button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the "Application" side-navigation item.
5. Fill in your motivation in the text area for your application.
6. Click on the "Submit" button.
7. Upon successful application submission, a confirmation dialog will appear.
8. Congratulations, you have now successfully applied to be a Tutor or TA!

3. Student with an Account and an Active Application

You have just successfully applied to be a Tutor or TA. There are no more actions for you to perform, and are simply awaiting approval. You may want to check the status of your application.

View Courses


1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Application” side-navigation item.
5. There, you will see confirmation text that informs you are currently awaiting approval.
6. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
7. If you are a first-time applicant and have never tutored or assisted via MyTutor before, you will see confirmation text that informs you that there are correctly no courses found for you. Otherwise, if you are a returning applicant, you will see buttons for each of the old courses that you have tutored or assisted for.
8. When you have been accepted to a course, a new blue button will appear under the “Courses” tab for that course, allowing you to click on it and perform your Tutoring/Assisting actions.

4. Administrator

As an Administrator on MyTutor, you have exclusive access to the “User Management” and “Course Management” views. There, you will be able to create, modify, and delete users and courses, respectively.

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on either the “User Management” or “Course Management” side-navigation as desired.
5. There, you will see a view that provides you with the ability to perform administrative user- or course-management actions. If you had chosen the “User Management” side-navigation item in Step 4, you will see a tab layout at the top which allows you to either manage other “Administrators”, “Employees”, or “Students”. Navigate to the correct tab, as desired.

Creating a User/Course

1. Click the “Create a ” button, which will cause a form dialog to appear.
2. Fill in the details for the entity you are creating.
3. Click the “Create” button.
4. Upon successful creation, a confirmation dialog will appear.
5. Congratulations, you have successfully created the new user/course.

Editing a User/Course

1. On the display grid, click on the row of the entity that you wish to modify, which will cause a form dialog that is populated with the selected entity’s details to appear.

2. Click the “Edit” button, which enables the modifying of editable fields.
3. Edit the details of the selected entity as required.
4. Click the “Save” button.
5. Upon successful edit, a confirmation dialog will appear.
6. Congratulations, you have successfully edited the selected user/course.

Deleting a User/Course

1. On the display grid, click on the row of the entity that you wish to delete, which will cause a form dialog that is populated with the selected entity’s details to appear.
2. Click the “Delete” button.
3. Upon successful deletion, a confirmation dialog will appear.
4. Congratulations, you have successfully deleted the selected user/course.

Resetting the System

You find yourself at the beginning of the semester/year, and all Students with MyTutor accounts need to have their application status reset back to idle, allowing them to apply and be considered in the current pool of applicants.

1. Ensure that you are on the “User Management” view, with the “Students” tab selected.
2. Click the “Reset System” button, which will display a user-confirmation dialog.
3. Confirm your action by clicking the “Confirm” button.
4. Upon successful system reset, a confirmation dialog will be displayed.
5. Congratulations, you have successfully reset the system!

Adding Employees to a Course

As an Administrator, you are responsible for inputting the relationship between an Employee (Course Convenor and Lecturer) and the courses of which they have worked for or are currently working for.

1. Ensure that you are on the “Course Management” view.
2. On the course display grid, select the row of the course that you wish to add employees to, which will cause a form dialog that is populated with the selected course’s details to appear.
3. Click the “Add Employees” button, which will cause another form dialog to show all employees currently registered on MyTutor.
4. Using the search filter as desired, select all the employees that you are tasked with adding to the course by using the check-box column.
5. Fill in the year for which these selected employees will be working for the selected course.
6. Click the “Add as Lecturers” or “Add as Course Convenors” button, depending on what role the selected list is to be or currently working for the selected course.

7. Upon successful addition, a confirmation dialog will appear.
8. Congratulations, you have successfully added the selected users as employees to the selected course, for an inputted year, as either Lectures or Course Convenors!

5. Course Convenor/Lecturer

As a Course Convenor or Lecturer for a course, one of your exclusive tasks is to accept TAs:

Accepting TAs

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
5. Select the course that you want to accept TAs for.
6. Click the “TA Listing” tab.
7. Click the “See Applications” button, which will open up a dialog that contains a grid of all the current applicants that have previously completed the course that you selected.
8. Clicking on each row in the applicant grid opens up a dialog that displays the selected user’s final grade that they got for the selected course, and their motivation for their current application.
9. Select all the TAs that you would like to accept by checking their check-box column.
10. Click the “Accept Applications” button.
11. Upon successful acceptance, a confirmation dialog will be displayed.
12. Congratulations, you have successfully accepted the students you selected as TAs for the current course.

6. Course Convenor/Lecturer & Accepted TA

As a Course Convenor, Lecturer, or TA for a course, you have the ability to perform the following:

Accepting Tutors

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
5. Select the course that you want to accept Tutors for.
6. Click the “Tutor Listing” tab.

7. Click the “See Applications” button, which will open up a dialog that contains a grid of all the current applicants that have previously completed the course that you selected.
8. Clicking on each row in the applicant grid opens up a dialog that displays the selected user’s final grade that they got for the selected course, and their motivation for their current application.
9. Select all the Tutors that you would like to accept by checking their check-box column.
10. Click the “Accept Applications” button.
11. Upon successful acceptance, a confirmation dialog will be displayed.
12. Congratulations, you have successfully accepted the students you selected as Tutors for the current course.

Creating or Modifying a Course’s Schedule

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
5. Select the course that you want to create/modify the Tutoring Schedule. This automatically defaults you to the “Schedule” tab.
6. Click the “Edit Schedule” button, which now allows you to interact with the Schedule Grid’s cells.

Creating a new Tutoring Session

1. Click the cell of the hour of when you want the new Tutoring Session to begin, and the correct column for the desired day, which will open a blank Tutoring Session form dialog.
2. Fill in the details of the new Tutoring Session.
3. Click the “Create” button to save your new TutoringSession to the local Schedule.
4. Click the “Save Schedule” button to save your new TutoringSession to the database.
5. Upon successful Schedule saving, a confirmation dialog is displayed.
6. Congratulations, you have successfully created a new Tutoring Session!

Update a Tutoring Session

1. Click the button of the Tutoring Session you want to modify, which will open a Tutoring Session form dialog populated with its existing details.
2. Make edits to the selected Tutoring Session by modifying form fields.
3. Click the “Save” button to save your modified TutoringSession to the local Schedule.
4. Click the “Save Schedule” button to save your modified TutoringSession to the database.

5. Upon successful Schedule saving, a confirmation dialog is displayed.
6. Congratulations, you have successfully edited an existing Tutoring Session!

Delete a Tutoring Session

1. Click the button of the Tutoring Session you want to delete, which will open a Tutoring Session form dialog populated with its existing details.
2. Click the “Delete” button to remove the selected Tutoring Session from the local Schedule.
3. Click the “Save Schedule” button to save the deletion of the selected Tutoring Session from the database.
4. Upon successful Schedule saving, a confirmation dialog is displayed.
5. Congratulations, you have successfully deleted an existing Tutoring Session!

Tutoring Statistics Viewing

To view the per Tutor Tutoring Statistic:

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
5. Select the course that you want to view the per Tutor Statistic for.
6. Click the “Tutor Listing” tab.
7. On the tutor display grid, clicking on a row will open up that Tutor’s tutoring attendance statistic., which provides a measure of that tutor’s tutoring experience.

To view the per Tutoring Session Statistic:

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
5. Select the course that you want to create/modify the Tutoring Schedule. This automatically defaults you to the “Schedule” tab.
6. Click the button of the Tutoring Session you want to view the statistics for, which will open a Tutoring Session form dialog populated with its existing details.
7. Click the “View Tutoring Statistics” button, which will open up a dialog that contains the selected Tutoring Session’s tutoring attendance statistics, which provides a measure of attendance to that specific session.

To view the per Course Tutoring Statistic:

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
7. Select the course that you want to view the per course Tutoring Statistics for.
8. Click the “Course Information” tab.
9. Click the “Get Tutoring Statistic” button, which will open a dialog that contains the selected course’s tutoring attendance statistics, which provides a measure of tutor attendance for that course.

10. Accepted Tutor

Sign Up to a Tutoring Session on a Course’s Schedule

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
5. Click the course of the Tutoring Session that you want to sign-up for, which will load the Schedule for that course by default.
6. Click on the Tutoring Session that you want to sign-up for on the Schedule grid, which will open a dialog with populated details about the selected Tutoring Session.
7. Click the “Sign Up” button.
8. Upon successful Tutoring Session sign-up, a confirmation dialog will be displayed.
9. Congratulations, you have successfully signed-up for a Tutoring Session.

Check-in to a Signed-Up Tutoring Session on a Course’s Schedule

You have just completed tutoring for a Tutoring Session that you are signed-up for.

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Courses” side-navigation item.
5. Click the course of the Tutoring Session that you want to check-in for, which will load the Schedule for that course by default.

6. Click on the Tutoring Session that you want to check-in for on the Schedule grid, which will open a dialog with populated details about the selected Tutoring Session.
7. If it is within the 15 minute grace period after the end time of the selected Tutoring Session, a button will be available for you to check-in. Click the “Check-In” button.
8. Upon successful Tutoring Session check-in, a confirmation dialog will be displayed.
9. Congratulations, you have successfully checked-in for your Tutoring Session.

11. General Instructions

The following are general instructions for performing various actions that may apply to numerous roles.

Editing your Profile

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “My Profile” side-navigation item.
5. Click the “Edit” button, which will enable the editing of your profile details.
6. Click the “Save” button.
7. Upon successful profile update, a confirmation dialog will be displayed.
8. Congratulations, you have successfully updated your profile!

Changing your Password

1. On the landing page, fill in your email and password.
2. Click on the “Sign In” button.
3. If the sign-in process is successful, you will be signed in and navigated to your dashboard.
4. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “My Profile” side-navigation item.
5. Click the “Change your Password” button.
6. Enter your new password in the appropriate field.
7. Write your new password in the confirmation field.
8. Click the “Save” button.
9. Upon successful password update, a confirmation dialog will be displayed.
10. Congratulations, you have successfully changed your password!

Signing Out

1. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the “Sign Out” side-navigation item.
2. Upon successful sign-out, you will be navigated back to the landing page.
3. Congratulations, you have successfully signed-out!

Switching to Dark Mode

1. On the side-navigation on the left side of your screen (which can be docked via the three-line icon in the header), click on the adjustable icon side-navigation item.
2. Clicking on it will toggle the theme of the screens as you desire, either light or dark.

This concludes the user manual for MyTutor software solution. If you encounter any issues or need further assistance, please contact our support team at mytutor.capstone@gmail.com!
