Task 1: Project Management

Team Structure and Teamwork Facility

The selected project management scheme for this coursework was the team leader model. The author of this document was chosen as the leader for the duration of the project. The team leader is responsible for coordinating tasks, tracking progress, uploading meeting notes, and reminding team members of deadlines and assigned tasks.

To facilitate the communication, the team created a WhatsApp group chat, allowing quick and efficient interaction among members. Additionally, the team setup a general GitHub repository to store and review group documents, along with individual repositories for personal contributions.

Documentation of Project Management

Date	Location	Attendees	Agenda
30 September, 2025. 12:00-13:00	Headington, AB115.	Malachai, Lucas, Ben, Tania.	Review coursework specification and case study.
7 October, 2025. 12:00-13:00	Headington, AB115	Malachai, Tania	Individual: quality requirements
14 October, 2025. 12:00-13:00	Headington, AB115.	Ben	Advance individual parts of coursework.
21 October, 2025. 12:00-13:00	Headington, AB115.	Ben, Tania	Complete tasks 2 and 3; commit on GitHub.
28 October, 2025.	Online	Tania, Ben, Lucas	Complete tasks 2 and 3; notify group members on progress.
4 November, 2025. 12:00-13:00	Headington, AB115.	Ben, Tania	Finish task 2 and 3, start working on task 4(a)
11 November, 2025. 12:00-13:00	Headington, AB115.	Ben, Tania	Contact Malachai for updates, continue working on task 4(a)
18 November, 2025. 12:00-13:00	Headington, AB115.	Ben, Tania	Finish task 4(a) and keep working on individual tasks.
25 November, 2025. 12:00-13:00	Headington, AB115.	Ben, Tania, Lucas	Complete task 4(b), update group members on progress.
27 November, 2025.	Online	Ben, Tania, Lucas	Continue with integration of subsystems, finalise task 5.

2 December, 2025.	Online	Ben, Tania, Lucas	Finalise Coursework, upload all files to GitHub (commit changes).

Allocation of Responsibilities

1) *USU Student App*: a mobile app to run on smart phones for students to participate in the student union's activities. Tania
2) *Student Union Management System*: a web-based application to run on desktop or tablet computer for officers of university-specific student union to manage and operate the student union. Malachai
3) *USU Operation System*: a web-based application to run on desktop or tablet computer for USU officers to manage the USU federation of student unions and realise its functions. Ben
4) *Society Leader App*: a mobile app to run on mobile phones and on desktop or tablet computers for society leader to organise and operator societies. Lucas

Report on Teamwork Performance: Individual

As team leader, my primary contribution to the coursework was ensuring that the group stayed on track and maintained steady progress toward our tasks. I facilitated communication by an easily accessible communication medium for updates, and I made myself a reliable point of contact by responding to messages and attending most meetings. One teammate was unresponsive for various weeks, so I reached out a few times to confirm whether he intended to complete his tasks When it seemed like he was not going to contribute, I prepared the group to move forward without his section, as it would not have affected our mark. Fortunately, he contacted the group shortly before submission deadline and informed us that he had an extension.

For technical contributions, I documented the content covered during our meetings to maintain a record of decision and progress. I also supported group members that needed clarification for their tasks. For example, I provided examples of the required diagrams and helped another member understanding the possible architectural approaches for the subsystems.

Collaboration withing the group was mixed but effective overall. Some tasks took group members longer than expected to complete, partly due to inconsistent communication at times. However, our weekly updates helped motivating everyone by reassuring that we were making progress. While all members completed their individual tasks, not all showed initiative beyond those. I used my leadership skills to address this issue by encouraging participation, coordination, and by pushing everyone to complete the project.

Task 2: Analysis and Specify Software Quality Requirements (USU Student App)

Software quality requirements are falsifiable statements which describe the features, functionalities, and constraints of a software system. Non-functional requirements describe the characteristics that the system has and how it works under certain conditions. The non-functional quality requirements for the USU Student App focus on identity and user verification (FR-ST-1) and event participation (FR-ST-6). This section of the report will discuss the quality requirements for FR-ST-1, Student Registration. The system must allow students to register to their university's student union with validation. The constraints discussed in the analysis are testable and their implementation should meet expected user needs.

Security and Privacy protection:

- The application must authenticate students via University SSO, before accessing the registration form.
- All personal data uploaded by students for registration (name, student ID, contact information, program information) must be encrypted with AES-256 during rest and TLS 1.3 during transit
- Passwords must be safely hashed and salted (with bcrypt or SHA)
- Users can request deletion or inspection of their personal data, as required under GDPR: Personal data must be stored and managed according to UK and EU data locality laws.
- Students must verify their identity with their university email to ensure they are members of the university.

Performance:

- Student registration form must load within 3 seconds.
- Students must be verified with University SSO within 5 seconds of submitting their registration.
- Input validation for missing fields or incorrect formatting must display a message within 2 seconds.
- The mobile app must respond even during peak periods to avoid delays during registration.
- The mobile app must display an loading indicator or progress indicator while processing (to improve user experience).

Reliability:

- The system must run 99.5% of the time, except for scheduled maintenance.
- Clear error messages must be displayed.
- If authentication with University SSO fails, the system must retry once before notifying the student of an error.
- User data must be backed up daily to secure cloud storage; after failure, it must restore to its previous working state.

Scalability:

- The system must support at least 20,000 concurrent users without any perceived decline in performance.
- The application must be designed and built on a scalable cloud infrastructure that supports horizontal scaling of servers.
- The system must facilitate integration with additional universities and student unions with minimal configuration changes.
- Registration data must be stored in a database that supports long-term growth of student records.

Task 3: Specification of Modelling Software Functional Requirements

The following section of the report presents and explains the Use Case Model and Activity Model designed for the student mobile application. The Use Case Model includes five actors: Student, Union Officer, Society Leader, University SSO, and Payment Service.

The system allows students to register to their university's student union, search for societies, join or quit societies, browse events, view event information, message societies, create new society requests, and register for events (including purchasing tickets when required). The University SSO verifies student identity during registration, ensuring that all users are members of a university in the UK. Society Leaders can message their societies and review messages sent by their members. Union Officers can approve new society requests created by students. The Payment Service deals with ticket purchases for specific events. The scope of the system focuses on functionality for students, as its design prioritises student experience, making the Union Officer and Society Leaders' roles limited.

The Activity Model focuses on the Registration to Student Union use case, as mentioned on the FR-ST-1 functional requirement. The activity starts when the student commences their registration, after which the system returns a Registration Form. Students then complete the registration form and the system processes the submitted data. Their identities are validated when the data is processed correctly and their student Id does not match others in the system. The registration is then confirmed, presenting students with options to finish the activity, view their stored information , or delete their information. This provides users with transparency and control over their personal data to fulfil UK data protection regulations and increase trust in the application.

Diagram 1.1: Use Case Model for USU Student App

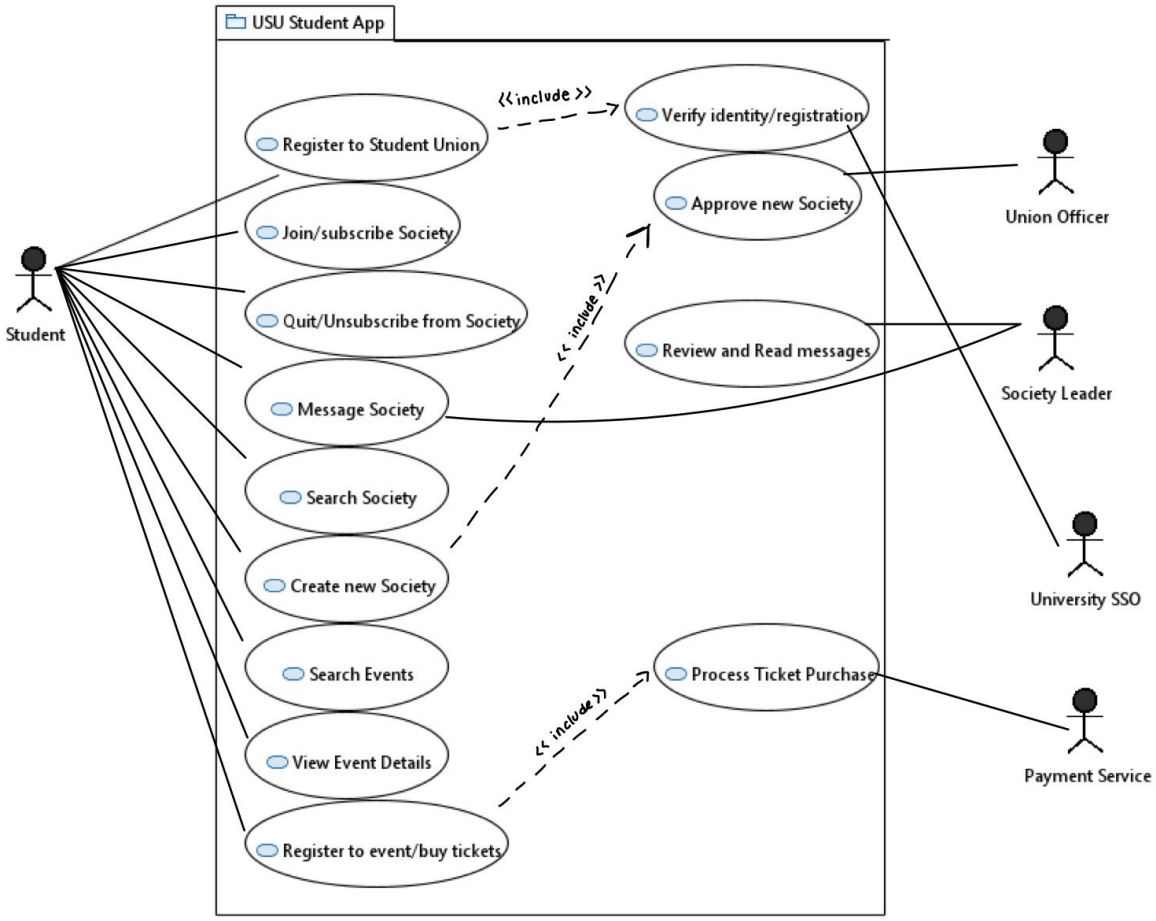

Diagram 1.1 illustrates how students interact with the system to register, join societies, access events, and communicate with societies. Due to a display issue in Papyrus, the associations

and relationships did not appear correctly in the exported diagram. To address this issue, the missing connections were manually drawn over the Papyrus Diagram.

Diagram 1.2: Activity Model for Student Registers to Student Union

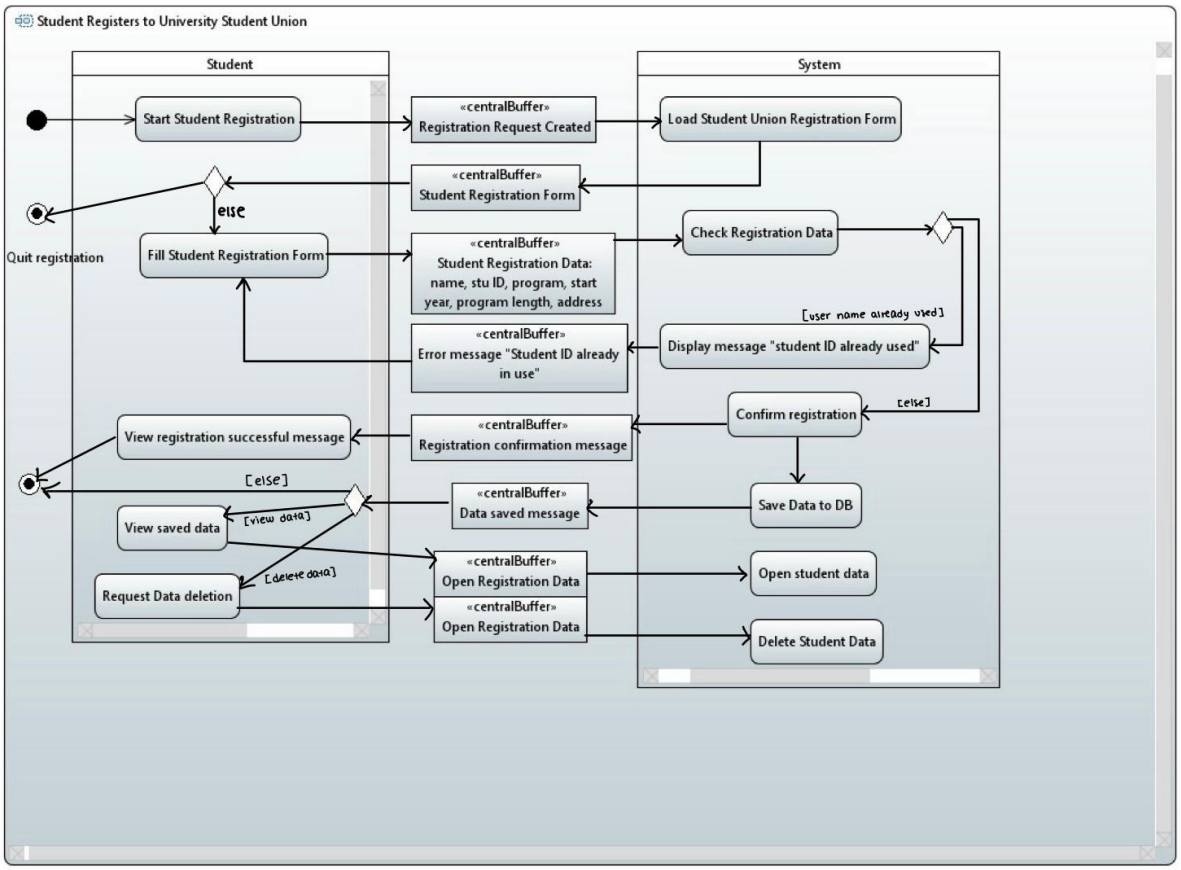

Diagram 1.2 shows the interactions between the student and the subsystem during the registration process for the Student Union. As with the previous diagram, Papyrus did not display the arrows, which were manually added to ensure accuracy in the design.

Task 4: Software Architectural Design

The following part of the report includes the architectural design of the USU Student Application, focusing on its microservices. It includes a description of its components and interfaces implemented in the subsystem's diagram.

Diagram 1.3: Architecture of USU Student App

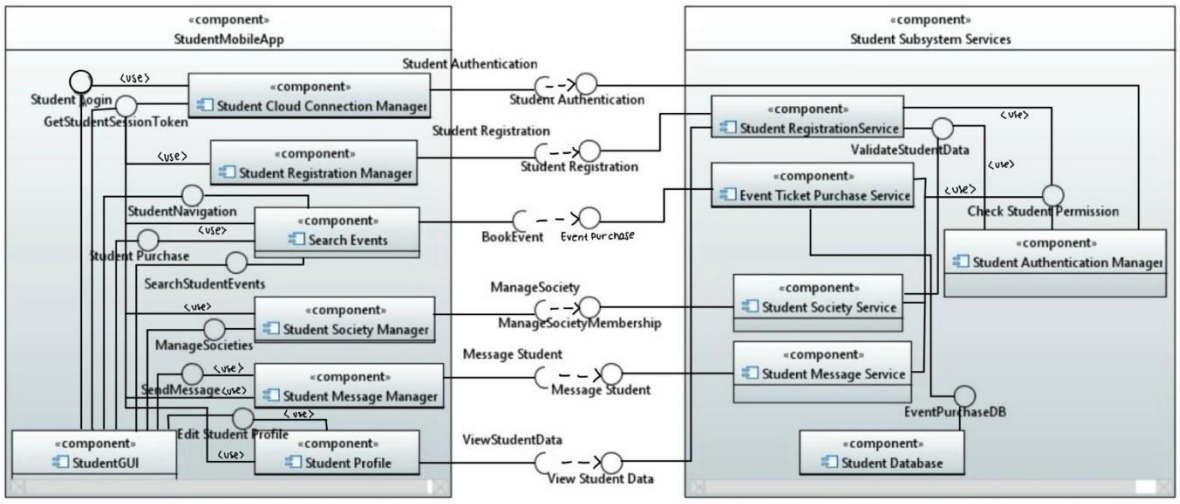

Diagram 1.3 displays the two-tiered component architecture of the USU Student Application. It includes components, interfaces (provided and required), and the connections between them. The two main components divide the subsystem between its client managers and cloud-based microservices. Papyrus did not show the connections and usages included in the diagram, being drawn manually as a consequence.

Component	Description	Stereo Type	Required Interfaces	Provided Interfaces
Student Mobile App	Student side application component which provides mobile interface to access all subsystem features.	Subsystem	StudentAuthentication, StudentRegistration, ViewStudentData, ManageSocietyMembership, MessageStudent, BookEvent	/
Student GUI	Provides user-interface (front-end interaction) for them to access system functionalities.	Process	/	StudentLogin, GetStudentSessionToken, EditStudentProfile, StudentPurchase, SearchStudentEvents, ManageSocieties, SendMessage, StudentNavigation

Student Cloud Connection Manager	Handles communication between the student mobile app and cloud service. Login requests are handles.	Process	StudentAuthentication	GetStudentSessionToken
Student Registration Manager	Handles the registration process and application data, and it validates student identity.	Process	Student Registration	/
Search Events	Allows the student to search and browse events, view their information, register, and purchase tickets.	Process	BookEvent	/
Student Society Manager	Manages operations that involve students and societies.	Process	ManageSocietyMembership	/
Student Message Manager	Handles communication (sending and receiving messages) between societies and student members.	Process	MessageStudent	/
Student Profile	Manages student profile data such as tudent Id, contact, course information, and other	Process	ViewStudentData	/

	personal information.			
Student Subsystem Services	Backend part of the subsystem running on the Cloud. Contains microservices for authentication, registration, society handling, messaging, and ticket purchasing.	Subsystem	/	StudentAuthentication, StudentRegistration, ViewStudentData, ManageSocietyMembership, MessageStudent, BookEvent
Student Registration Service	Back end microservice which handles new student registration applications, manages user data, and validates their identity.	Service	/	StudentRegistration, ViewStudentData, validateStudentData
Event Ticket Purchase Service	Microservice which handles event registration and ticket purchases for students. Payments are processed and ticket records are stored in the database.	Service	EventPurchaseDB	BookEvent
Student Authentication Manager	Microservice for validating student authentication and login credentials. Checks student permission for various operations.	Service	/	StudentAuthentication, CheckStudentPermission

Student Message Service	Microservice which processes and broadcasts student messages exchanged within society chats.	Service	/	MessageStudent
Student Society Service	Microservice which manages student society memberships. It processes new society applications and it manages requests to join or quit societies.	Service	/	ManageSocietyMembership
Student Database	Database service which stores and manages student profiles, registrations, ticket purchase, and society data.	Entity	/	EventPurchaseDB

Interfaces:

Name	Student Authentication	
Provider	Student Authentication Manager	
Operation	Signature	studentLogin(studentId: string, password: string, ipAddress: IPAddress): sessionToken: String
	Function	Check if student password matches the stored password. Complete login if successful and return sessionToken (string with student data and their university). If login not successful, return error message.

Operation	Signature	validateStudentSession(sessionToken: String): Boolean
	Function	Check the sessionToken to verify if it is active or if it has expired. If valid, return true.

Name	Student Registration	
Provider	Student Registration Service	
Operation	Signature	registerStudent(name: String, studentId: String, studyProgram: String, yearOfStart: Integer, lengthOfProgram: Integer, contactAddresses: ContactInfo, supportingDocuments: DocumentList): registrationResult: RegistrationStatus
	Function	Processes student requests to join their university's student union. Registration data is validated, and if successful, student receives user access with a membership Id. If validation unsuccessful, it returns a fail message.
Operation	Signature	validateUniversity(universityId: String): Boolean
	Function	Checks that the student's university is part of USU, returns true if the University is a valid member. Otherwise, returns false to ensure only University students can access.

Name	View Student Data	
Provider	Student Registration Service	
Operation	Signature	getStudentProfile(studentId: String, sessionToken: String): StudentProfile
	Function	Retrieves name, student Id, study program, year of start, length of program, contact address, and USU membership Id. Student's identity is validated with sessionToken to ensure the student has access to view the information. If they have access, it returns a StudentProfile object, otherwise, it returns an error message.
Operation	Signature	updateStudentProfile(studentId: String, sessionToken: String, updatedProfile: StudentProfile): updateStatus: Boolean
	Function	Student profile data can be updated with new information. Student's identity is verified with sessionToken to ensure they have access to modify the profile. Return update if successful or error message if it fails.

Name	Manage Society Membership	
Provider	Student Society Service	
Operation	Signature	searchSocieties(studentId: String, sessionToken: String, searchCriteria: String): SocietyList
	Function	Seraches for University specific societies based on the search criteria. It returns a list of societies that match the search.
Operation	Signature	joinSociety(studentId: String, sessionToken: String, societyId: String): membershipStatus: Boolean
	Function	Process student requests to join societies. If successful, student becomes a member of the society, otherwise, returns false and an error message.
Operation	Signature	quitSociety(studentId: String, sessionToken: String, societyId: String): quitStatus: Boolean
	Function	Process student requests to quit societies. If successful, student stops being a member of the society, otherwise, returns false and an error message.
Operation	Signature	subscribeSociety(studentId: String, sessionToken: String, societyId: String): subscriptionStatus: Boolean
	Function	Processes subscriptions to societies (event and society notifications become available). If subscription is successful, return true, otherwise, returns false.
Operation	Signature	unsubscribeSociety(studentId: String, sessionToken: String, societyId: String): unsubscribeStatus: Boolean
	Function	Unsubscribes student from societies (stop receiving updates about the society and events). If unsubscription successful, return true, otherwise, return false and error message.
Operation	Signature	createNewSociety(studentId: String, sessionToken: String, societyApplication: SocietyApplicationData): applicationResult: ApplicationStatus
	Function	Processes student applications (manually by union Officer) to create a new society. Student identity is verified. If application successful, they become society Leader and their application status appears as successful.

Name	Message Student
Provider	Student Message Service

Operation	Signature	sendMessageToSociety(studentrId: String, sessionToken: String, societyId: String, messageContent: String): messageStatus: Boolean
	Function	Students can send messages to a society they are members of. Student identity and society membership is verified, and if validation is successful, their messages are broadcasted to the other society members.

Name	Book Event	
Provider	Event Ticket Purchase Service	
Operation	Signature	addEvent(studentId: String, eventId:String): Event
	Function	Adds events to students' event database. If successful, the event get added to their future events. If payment is required, they will be redirected to the payment process. Otherwise, returns an error message.
Operation	Signature	getBookingDetails(studentId: String): EventDatabase: <list>
	Function	Retrieves information from event database and displays event data to user. If unsuccessful, returns error message.
Operation	Signature	cancelBooking(studentId: String, EventId: String): bookingSatus
	Function	Cancels event bookings and processes refund If necessary. If successful, event booking is removed from student's record, otherwise, returns error message.

Name	Event Purchase	
Provider	Event Ticket Purchase Service	
Operation	Signature	refundTicket(transactionId: String): String
	Function	Processes ticket refund. If successful, event booking is cancelled and user receives a refund.
Operation	Signature	purchaseTicket(studentId: String, eventId: String): String
	Function	Processes ticket purchase. If successful, event gets added to their Event Database.

Name	Student Login

Provider	/	
Operation	Signature	displayLoginForm(): void
	Function	Login form is displayed for students to fill with their studentId and password. Once authenticated properly, they can access the system.

Name	Get Student Session Token	
Provider	Student Cloud Connection Manager	
Operation	Signature	getSessionToken(): String
	Function	Returns the active sessionToken the student is using for authentication. Returns an empty string if there is no active session.

Name	Edit Student Profile	
Provider	/	
Operation	Signature	displayProfileEditForm(currentProfile: StudentProfile): void
	Function	Student profile (with their data) is displayed and ready to be modified by the user.

Name	Student Purchase	
Provider	/	
Operation	Signature	processPayment(amount: Decimal, eventId: String, ticketQuantity: Integer): PaymentStatus
	Function	Handles event ticket purchases, validates payment information and processes the transaction. Returns status of payment if successful or failed.

Name	Search Student Events	
Provider	/	
Operation	Signature	displayEventSearchInterface(): void
	Function	Displays event search interface for students. Students can enter criteria and view a list of available events.

Name	Manage Societies	
Provider	/	
Operation	Signature	displaySocietyManagementInterface(): void
	Function	Displays the society interface for student management (join, quit, subscribe, unsubscribe) and for new society applications.

Name	Send Message	
Provider	/	
Operation	Signature	displayMessageWrite(societyId: String): void
	Function	Displays user interface where students can write and send messages to the societies they are current members of.

Name	Student Navigation	
Provider	/	
Operation	Signature	navigateToScreen(screenName: String): void
	Function	Handles student navigation between sections of the application.

Name	Check Student Permission	
Provider	Student Authentication Manager	
Operation	Signature	checkPermission(sessionToken: String, requestedOperation: String, resourceId: String): Boolean
	Function	Verifies that the student has permission to perform an operation on the system based on their sessionToken. Student information is extracted, if they have permission it returns true.
Operation	Signature	validateStudentMembership(studentId: String, societyId: String): Boolean
	Function	Verifies if a student is a member of a specific society. Returns true if members, otherwise, returns false.

Name	Validate Student Data	
Provider	Student Registration Service	
Operation	Signature	validateStudentCredentials(studentId: String, universityId: String): ValidationResult
	Function	Student credentials are validated once supporting documents have been checked by the system. Returns ValidationResult with the student's enrolment status.

Name	Event Purchase DB	
Provider	Student Database	
Operation	Signature	updateEvent(event: String, updatedEventData: EventData): void()
	Function	Updates event information from the database with new data.
Operation	Signature	getEvent(eventId: String): Event ()
	Function	Retrieves event information from the database. It returns an Event containing all event information.
Operation	Signature	saveTransaction(transaction: Transaction): void
	Function	Saves specific event transaction information in the database.
Operation	Signature	getTransaction(transactionId: String): Transaction
	Function	Retrieves transaction details from the database. It returns a Transaction with all of its information.
Operation	Signature	saveBooking(student: Student, event: Event, ticketCount: int): void()
	Function	Saves an event booking in the student database to keep a record of events they are registered to.

Task 5: Software Detailed Design

This part of the report, and final one, includes a structural model and behaviour model of the event ticket purchase service in the subsystem architecture.

Diagram 1.4: Structural Model of Event Ticket Purchase Service

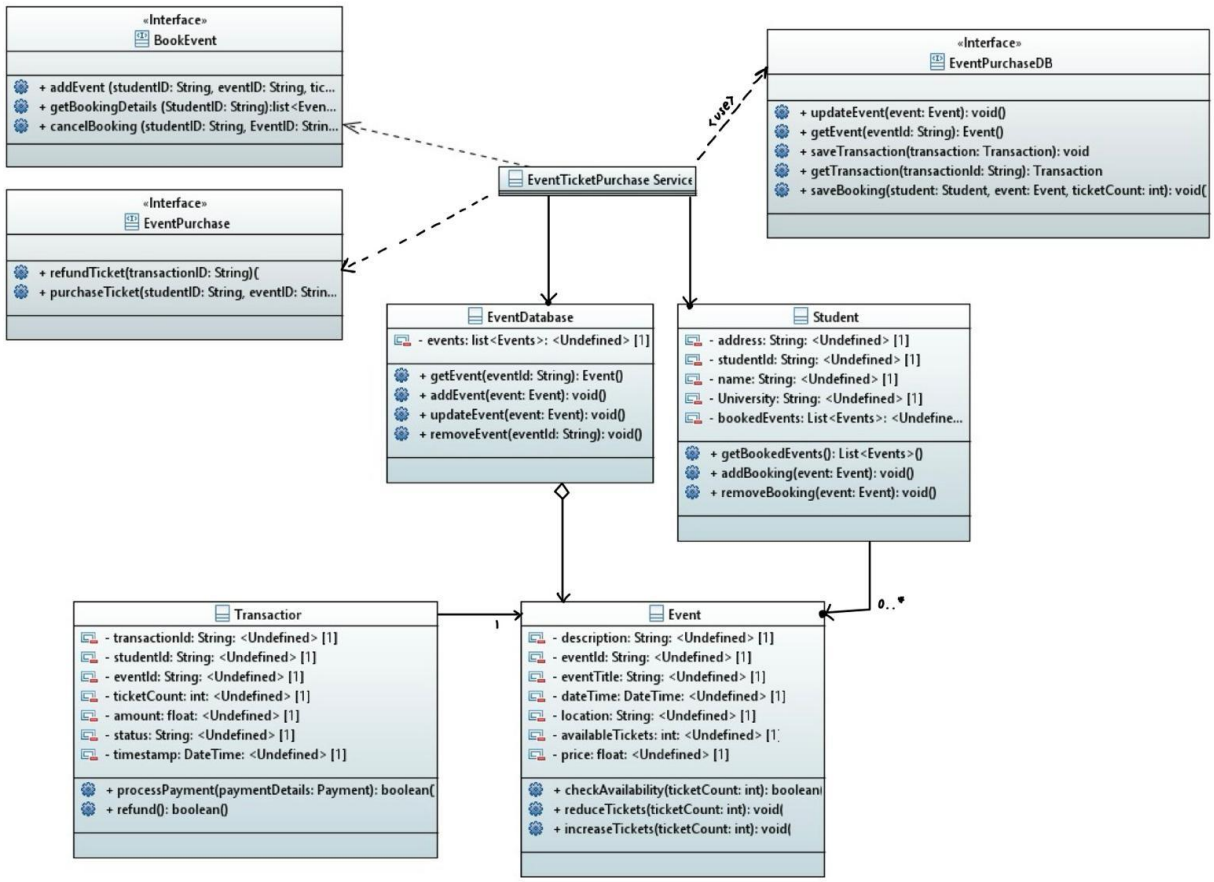

Diagram 1.4 displays the class diagram designed for the Event Ticket Purchase Service component. It includes three interfaces from the previous section (BookEvent, EventPurchase, and EventPurchaseDB) and EventDatabase, Student, Transaction, and Event classes. The arrows in the diagram were manually drawn due to complications with Papyrus.

Diagram 1.5: Event Ticket Purchase Behaviour Model

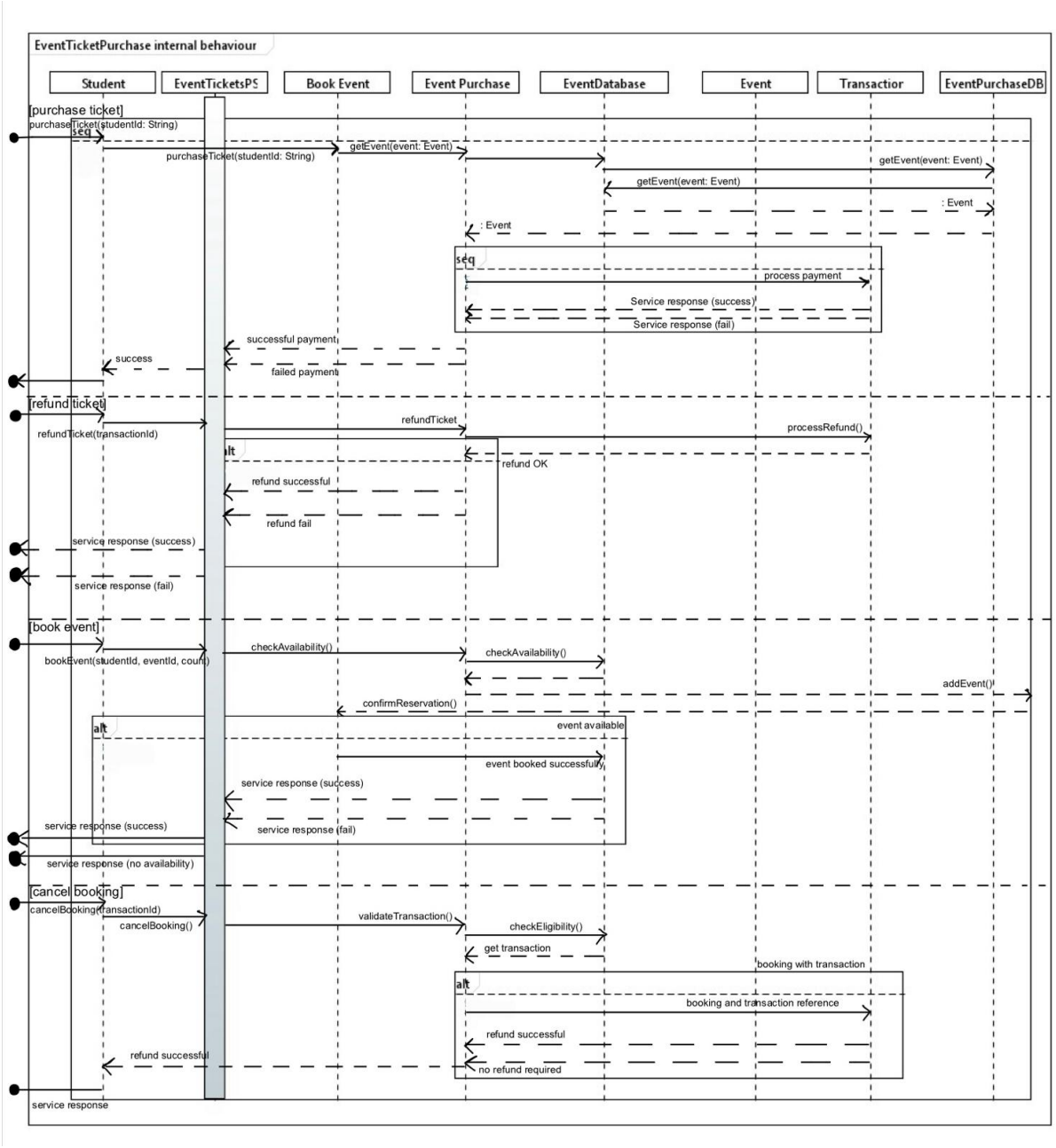

Diagram 1.5 is a sequence diagram for the Event Ticket Purchase Service component. The main actions covered by the diagram are purchase ticket, refund ticket, book event, and cancel booking.

References

‘Component’ (2016) UML Diagrams. Available at: https://www.uml-diagrams.org/component.html (Accessed: 4 December 2025)

Decinco, J. (2024) Understanding Cryptography and AES-256: A Deep Dive. Available at: https://medium.com/@jeromedecinco/understanding-cryptography-and-aes-256-a-deep-dive-d9e374272753 (Accessed: 18 October 2025)

Frontegg (2024) What Is User Authentication? A 2025 Guide for Modern Apps. Available at: https://frontegg.com/blog/authentication (Accessed: 18 October 2025)

Honcharuk, Y. (2024) How To Write A List of Requirements For A New Application. Available at: https://dashdevs.com/blog/how-to-write-good-software-requirements-specification-for-your-mobile-application/ (Accessed 14 October 2025)

Hong Zhu (2025) ‘Lecture 10’. *COMP5047: Applied Software Engineering*. Oxford Brookes University. Unpublished.

Krüger, G. (2025) Non-Functional Requirements: Tips, Tools, and Examples. Available at: https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples (Accessed 14 October 2025)

Mistele, K. (2020) How to securely hash and store passwords in your application. Available at: https://medium.com/codelighthouse/how-to-securely-hash-and-store-passwords-in-your-next-application-edf15873a432 (Accessed: 18 October 2025)

Satpathy, A. (2024) How to Write Good Software Requirements (with Examples). Available at: https://www.modernrequirements.com/blogs/good-software-requirements/ (Accessed: 14 October 2025)

Vaquero, L., Rodero-Merino, L. and Buyya, R. (2011) Dynamically scaling applications in the cloud. Available at: https://dl.acm.org/doi/abs/10.1145/1925861.1925869?casa_token=xe9HDXySDu4AAAAA:JKyTf_GoQdQgEBLYohxxBQJ7nByhsLlvtYI-lEFDV7oWlCWxEwA0U-PsbLK0_1DOVaAEdLgA-i5C (Accessed: 18 October 2025)