# CS 255 System Design Document Template

This template lays out all the different sections that you need to complete for Project Two. Each section has guidance to prompt your thinking. You will need to continually reference the interview transcript as you work to make sure that you are addressing your client’s needs. There is no required length for the final document. Instead the goal is to complete each section based on what your client’s needs are. Remove this note when you are finished, and replace all bracketed text with the relevant information.

## UML Diagrams

### UML Use Case Diagram

*[In Module Six, you were asked to complete a use case diagram based on your system design. If you would like to make any adjustments to your diagram, please do so. Please insert your use case diagram here. Check to make sure that you included appropriate components and symbols and that your design meets the client’s needs.]*

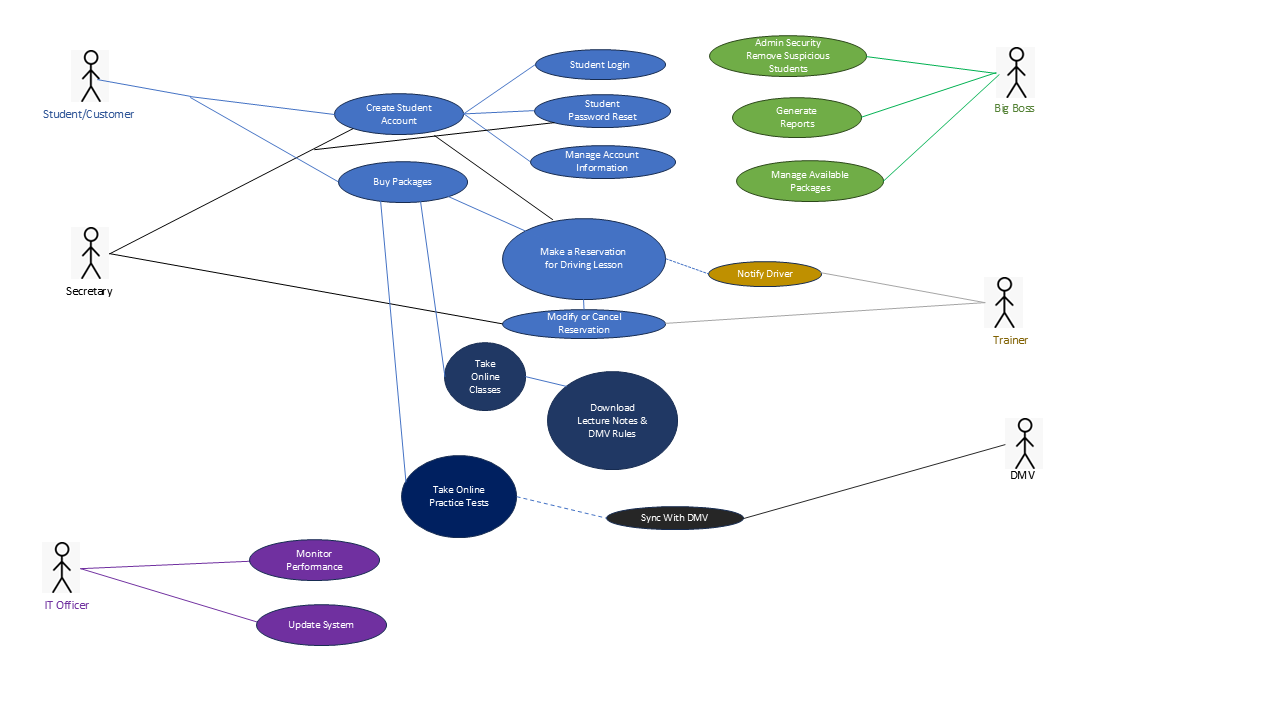
A Use Case Diagram is a UML diagram that visually represents the illustrations and interactions between users (actors) and a system, or the different moving parts within a system. This can include other actors such as the admins, the driving instructors, and even the DMV and System Security. It’s typically easier to digest information for non-technical audiences to illustrate these relationships between functionalities and users. They’re crucial for definition system scope and requirements. (srinam. 2024)

Actors:

* **Customer:** Aka the student for DriverPass, they have the ability to book, modify, or cancel a driving lesson mainly. They can create their account, personalize the account, take practice tests, and download notes.
* **Secretary:** Minor role, they answer phone calls, they also manage reservations if needed.
* **It Officer:** Manages the system maintenance and monitors the performance. The interview didn’t really go in depth on how, but I imagine they look up run times on specific areas and review feedback from the customer in high traffic areas.
* **Owner:** Generates reports of system activity, can modify availability of packages, and edit user roles / accessibility.
* **System:** DriverPass system, connects to the DMV to keep practice tests up to date, along with downloadable notes and DMC regulations.
* **DMV:** external state regulated agency. They receive their regulations by the state and have to update tests accordingly.
* **Drivers/Instructors:** Takes part in the lessons, receives notifications of driving lessons, potentially might have the option to cancel and reschedule lessons.

**Use-Cases:**

* **Create Student Account:** Students can input their personal information such as email address, and password to provide a personal identity to match with their login ID. We can also include other identity options such as a 2factor identifier
  + **Student Login:** Customers can then log into their account or log out.
  + **Student password reset:** Customers can also reset their password by providing their 2factor identification, providing their email, or answering a security question.
  + **Manage Account Information:** Customers update their personal information, such as changing their username, updating their profile picture, or even changing their save payment options, or billing address.
* **Admin Security Options:** Admins can Remove or modify users within the system, allowing for enhanced security within said system. For example, if an admin detects a suspicious student performing a breach obtaining online courses without paying within their reports, they can ban or remove this user from the system through their IP address. On the other hand, password resets will also be given as an option to both IT Officers & Admins, for students who may potentially forget their password and require assistance with retrieving their account.
  + **Generate Reports:** The owner/admins can generate activity reports to monitor reservations and changes. This way the admins can detect if someone is scheduling lessons without paying, or if sensitive information is being taken or viewed within the system. Breaches can be detected or even changes to the system, thus the owner will have these reports to better locate them.
    - **Manage Packages:** The owner can enable or disable specific driving lesson packages, or even add different package types, such as customizing the price or the hours of availability to enhance flexibility with the market demand.
      * **Buy Packages:** Depending on what’s available the customer can purchase a package or lesson available.
* **Make a reservation for driving lesson:** Customers book driving lessons, selecting the time, date and trainer.
  + **Notify Driver:** Once a reservation is made, the driving instructor is notified in advance to show up to the driving spot.
  + **Modify or Cancel Reservation:** Customers or the secretary or the trainer can modify or cancel existing appointments.
* **Take Online Classes:** Customers can access learning materials such as online video-oriented lectures.
  + **Download Lecture Notes:** An added feature. Students can download lecture notes to better prepare for practice tests. These notes are provided in PDFs that offer OCRs for easier flexibility in utilizing the information provided within said notes. OCRs allow for users to easily be able to copy and paste texts, thus speed readers such as Balabolka or Grammarly can provide better retention of information.
  + **Take Online Tests:** Customers can download a simulation tool (exe). This is an interactive platform that can be run locally on any computer without internet. However, downloading the initial .exe will require internet, the student can take these practice tests anywhere through the simulation tool. These practice tests grade the student’s responses and provide necessary feedback to each of their answers.
  + **Handling Payments:** There will be a history tracker for every transaction made within DriverPass. This is done to track profits and allow the company to distinguish if the system is financially sustainable. Each transaction will have an account ID associated with it, so that Customers can also view their own transaction history for their own personal use, such as tracking their spending or confirming if their transaction actually went through or not.
* **Sync with DMV:** Updates the practice tests and notes with new questions and regulations in accordance to what the DMV wants to focus on.
* **Monitor Performance:** Primarily done by the IT officer, they will review areas with high complaints and significant performance degradation. It is important to note that the IT officer should not be confused with the cloud-based infrastructure (such as AWS or Azure), which handles hardware related issues. The IT officer is responsible for addressing software-related issues such as fixing newly found bugs, implementing patches to fix data breaches, or updating the UI in places with high user complaints, etc.

**

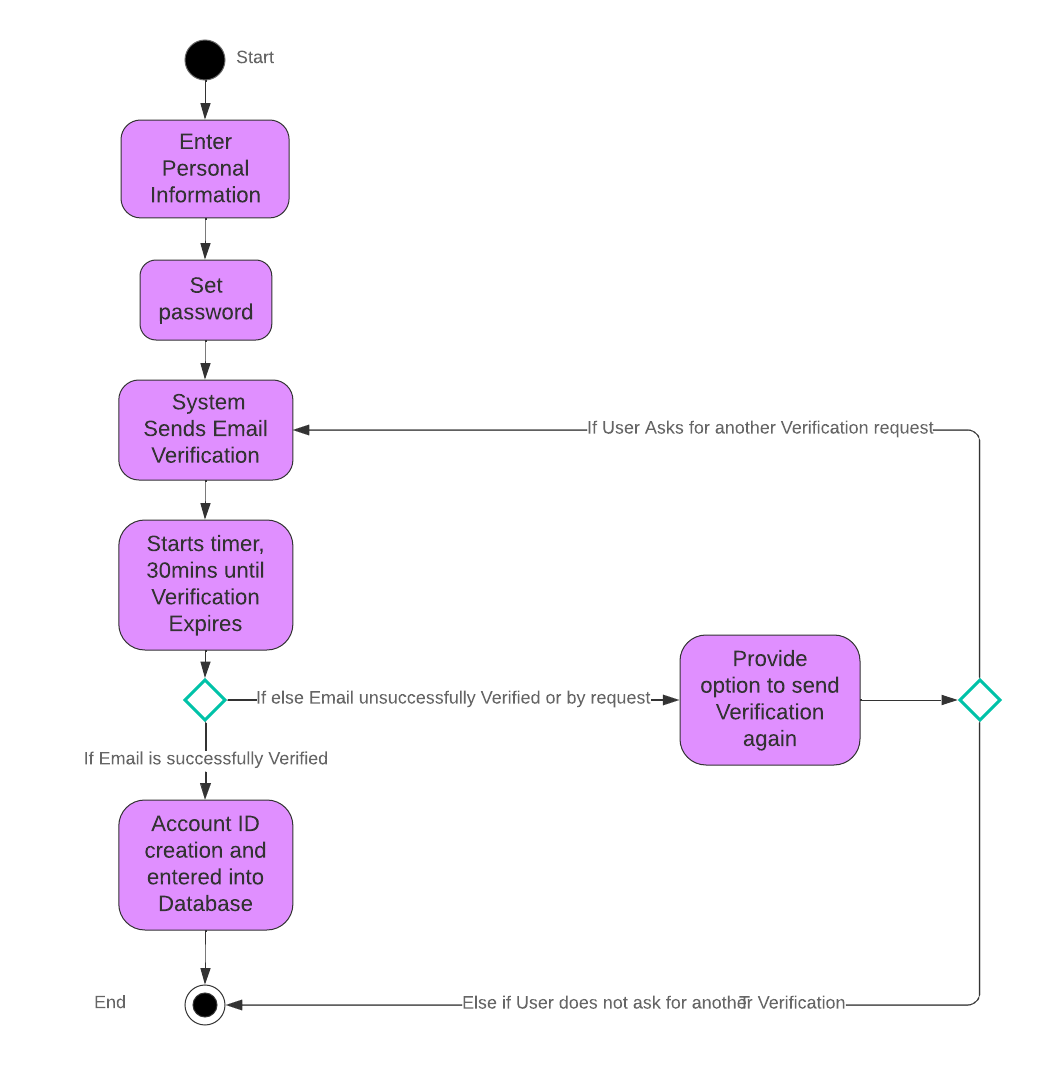
### UML Activity Diagrams

*[You were asked to choose* ***two*** *use cases and create* ***two*** *activity diagrams, one for each use case. Please insert* ***both*** *of your activity diagrams here. Check to make sure that you included appropriate components and symbols and that your design meets the client’s needs.]*

Below are two examples of activity diagrams to illustrate the specific steps of some of the use cases. I’ll only be including two in this template (Creating an Account & Making a Reservation for a Driving Lesson), but individual lists and diagrams will be included for the rest of the use-case in separate packets. These two are just examples of what to expect.

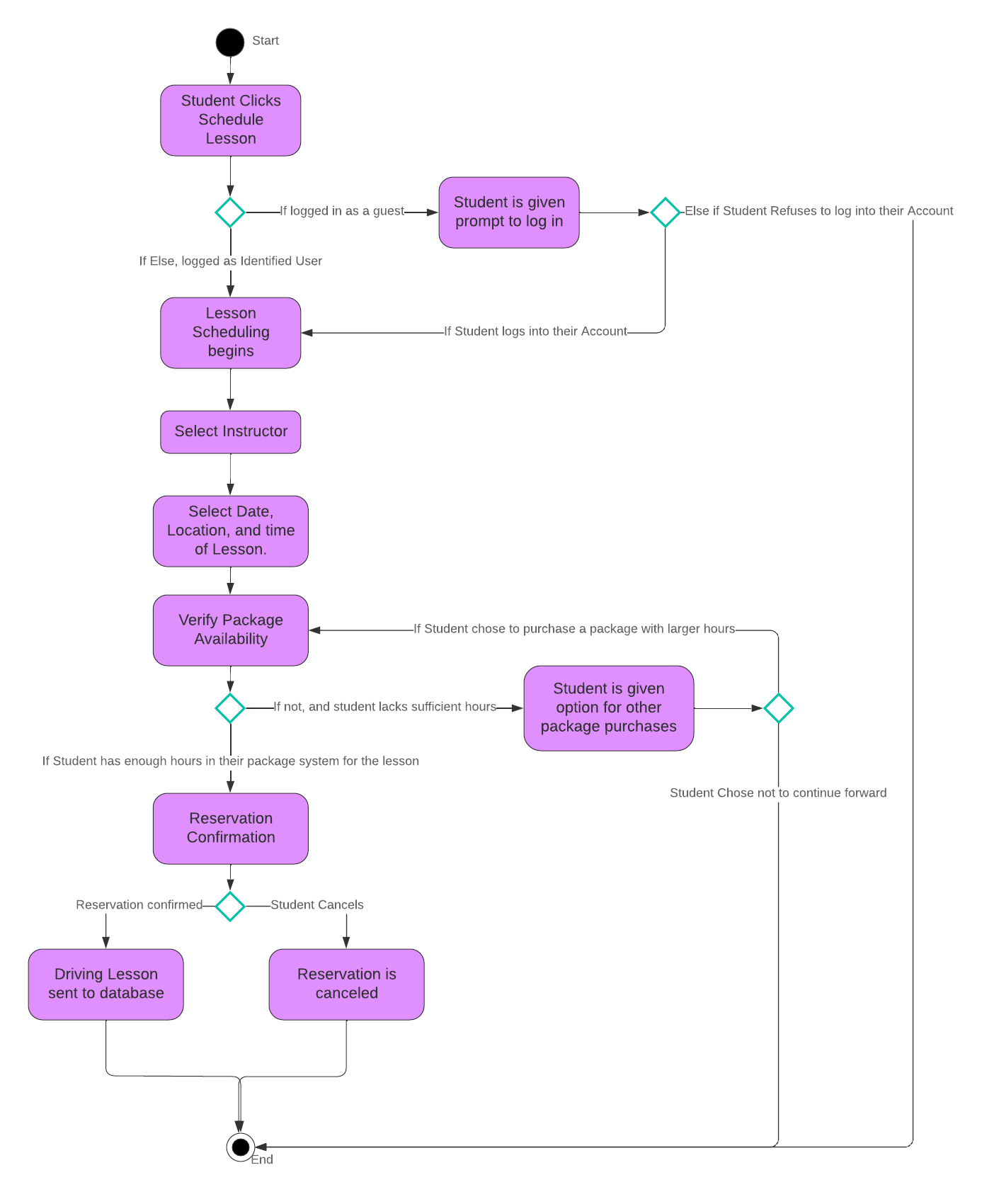
**Creating an Account:**

* **Start:** The process begins. Account creation typically starts with a large button on the top right if the user enters the website as a guest from an unrecognized IP address. There will be an option to create an account rather than an option to log in on the top right of the main dashboard.
* **Enter Personal Information:** Customer/Student will provide their personal details such as email (Required), address, username, personal photos, special needs if any.
* **Set Password:** Students will then be prompted to create a password to keep their account secure.
* **Verify Email:** A verification link will be provided to the customer’s email address. This is to ensure user identity and ownership of the specific email. There will also be a timer signaling a deadline for the link to expire within 30 minutes. If the user fails to answer the link within the deadline, or if other internet issues occur, the user will have the option to send in another request.
* **Decision (Diamond 1):** Did the email successfully get verified in time?
  + **Yes:** Move forward to account creation
  + **No:** Provide option to send email again
    - **Decision (Diamond 2):** Did the user ask for another verification?
      * **Yes:** Sen another email Verification and loop back.
      * **No:** End the diagram.
* **Create Account in System:** The system gives the student a unique user-ID that is attached to their personal email and username, and the account is provided to the database.
* **End:** The process ends.



Making a Reservation for a Driving Lesson:

* **Start:** The process begins.
* **Student clicks ‘Schedule Lesson’:** The option is provided on the main dashboard as one of the main options.
  + **Decision (diamond 1): Verify if student is logged in:**
    - **No:** if the student is not logged into their account, a prompt for them to log in will be provided, and it will not allow them to continue unless their identification is confirmed. (diamond 2)
    - **Yes:** If the student is logged in, the system will continue to the process of scheduling a lesson.
* **Select Instructor:** The student will choose from the available instructors.
* **Select Date, Location, and Time:** The student will then choose from the calendar of the available dates and times. All unavailable times will not show up as options, along with the number of hours they wish to practice.
* **Decision: (diamond 3): Verify Package & Hours of Availability**
  + **Yes:** If the student has adequate hours with their package the system will continue with the reservation process. The package system is a separate system that provides the students with a limited amount of lesson hours. There are several different package options for students to choose from.
  + **No:** If not, the student will be provided with a message stating they have insufficient hours, and will be given the option to purchase a different package.
* **Reservation Confirmation: (diamond 4)** 
  + **Yes:** If the student does a final confirmation, the driving lesson is sent to the database, the instructor is given a notification, and the student receives a receipt and confirmed email.
  + **No:** If not, the reservation is canceled and the process ends.
* **End:** the process is complete.



### UML Sequence Diagram

*[You were asked to create a sequence diagram based on* ***one*** *of the use cases you chose. Please insert your sequence diagram here. Check to make sure that you included appropriate components and symbols and that your design meets the client’s needs.]*

A Sequence Diagram is a UML based diagram used to visualize the interaction between objects in a sequential order. It focuses on how objects communicate with each other over time. (GeeksforGeeks. 2024) They illustrate object interactions, message flows, and the sequence of operations, making them valuable for understanding use cases, designing system architecture and documenting complex processes. I’ll be using use-case of a customer’s account creation as an example.

Actors:

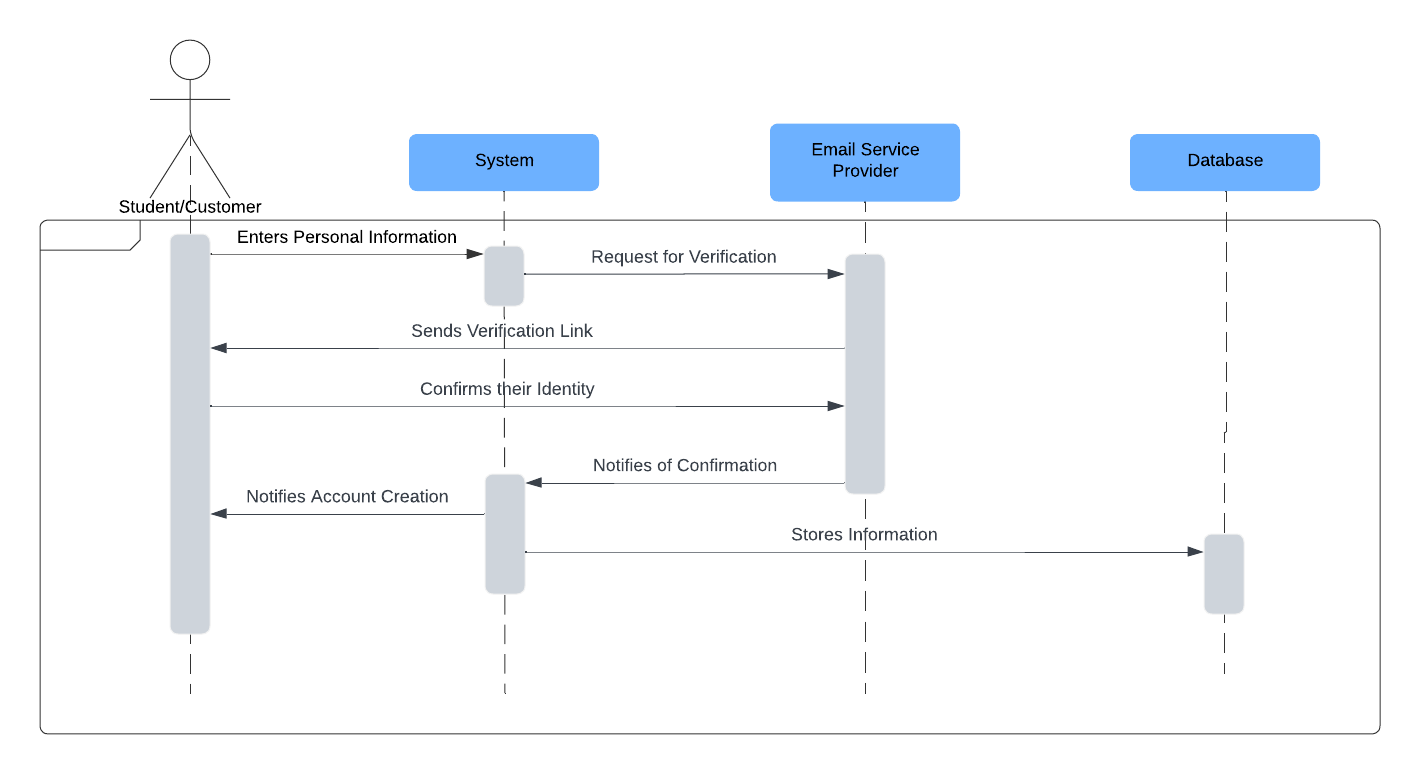
* Student/Customer
* System(DriverPass)
* Database(Cloud Infrastructure: EX: Azure/AWS.)
* Email Service Providers (API for Google, Yahoo, or Outlook)

Steps:

* **Student -> System:** Enters personal information (name, email, password, etc) and submits it.
* **System (DriverPass)-> Email Service (API):** The system sends a request to the email service (Google, Yahoo, Outlook) to send a verification email to the student
* **Email Service (API) -> Student:** The email service sends the verification email with a link
* **Student -> Email Service (API):** The student clicks the verification link to confirm their identity
* **Email Service (API) -> System (DriverPass):** The email service notifies the system that the student’s identity has been validated.
* **System (DriverPass) -> Student:** The system gives the student a notification that their account has been successfully created
* **System (DriverPass) -> Database (Cloud – Azure/AWS):** The system assigns the student’s account information with a unique ID, and stores it into the third-party database.

Summary

* **Enters Personal Information:** The **Student** starts the process by creating their account. They interact with the **System**, by clicking the “create the account” button, and entering their account information. (name, email, password).
* **Request for Verification:** The **System** notifies the **Email Service** (Google, Yahoo, or Outlook API) to request the sending of a verification email to the student’s email address
* **Sends Verification Link:** The **Student** receives the verification email from the **Email Service**
* **Confirms Identity:** The student clicks the link and validates their identity.
* **Notifies of Confirmation:** The **Email Service** notifies the **System** that the student’s identity has been successfully verified, and the system assigns the student’s information with a unique ID.
* **Notifies Account Creation:** The **System** informs the **Student** that their account has successfully been created
* **Stores Information:** Finally, the **System** stores the student’s information into the **Database (**Cloud infrastructure: Azure or AWS) and assigns a unique ID to their account.



### UML Class Diagram

*[You were asked to create a class diagram based on the different classes and attributes needed for your system design. You are* ***not*** *required to include methods, but you may if you wish. Please insert your class diagram here. Check to make sure that you included appropriate components and symbols and that your design meets the client’s requirements.]*

**Student Class:**

Attributes:

* studentId: int – unique identifier for each student
* username: String – The student’s username
* email: String – The student’s email address
* password: String – The student’s password for security
* phone: String – Contact number for the student
* address: String – The student’s IP Address
* packageSelected: String – The driving lesson package chosen by the student
* lessonHoursRemaining: int – The number of driving lesson hours remaining for the student
* accountStatus: String – The status of the student’s account (Active or suspended)
* paymentDetails: String – Payment information stored for transaction history

Methods:

* createAccount(): void – Creates a new account for the student by capturing their personal information
* login(): void – Authenticates the student using their username and password
* resetPasword(): void – Allows the student to reset their password if they forgot it.
* bookLesson(instructorID, lessonDate): void – Allows the student to schedule a driving lesson with a specific instructor
* modifyLesson(): void – Enables the student to modify an existing lesson’s time or date
* cancelLesson(): void – Allows the student to cancel a scheduled leson.
* viewProgress(): void – Displays the student’s progress in the lessons, including hours completed and test scores.
* payForPackage(packageID): void - Facilitates payment for the selected driving lesson package.
* updateProfile(): void - Allows the student to update personal details such as phone number or payment options.
* downloadMaterials(): void - provides access to downloadable study materials, such as lecture notes or practice test content.

**Admin Class:**

Attributes:

* adminID: int – a unique identifier for each admin
* username: String – Admin’s username for login
* email: String – Admin’s email address
* role String - The role assigned to the admin (System Admins or IT Officer)
* activityLog(List<String>) - A record of the admin’s activities in the system, such as password resets, user modifications, system modifications.

Methods

* resetStudentPassword(): void – Resets the password for a specified student upon reqest
* editUserRole(): void – edits the user role to be promoted to admin, instructor, or IT Officer.
* removeUser(): void – Removes or disables a user from the system.
* generateReport(): void – Generates system reports such as activity logs, user statistics, or financial summaries.
* managePackages(): void – Enables the admin to add, remove, or disable driving lesson packages.

Instructor Class:

Attributes:

* instructorID: int – a unique identifier for each instructor
* username: String – The instructor’s username
* email: String – The instructor’s email address
* phone: String – The instructor’s contact phone number
* assignedLessons: List<Lesson> - A list of lessons assigned to the instructor via student appointments.
* availability: boolean – indicates if the instructor is available for lessons.
* vehicleAssigned: String – The vehicle assigned to the instructor for driving lessons.

Methods:

* viewSchedule(): List<Lesson> - Retrieves the instructor’s schedule of assigned lessons
* modifyLessonSchedule(lessonID): void – allows the instructor to modify a lesson’s time or date, subject to availability
* cancelLesson(lessonID): void – Cancels a lesson that has already been scheduled
* markLessonComplete(lessonID: void – Marks a lesson as complete after its conducted, providing feedback if necessary
* updateAvailability(status): void – Updates the instructor’s availability status.
* assignVehicle(vehicleID): - Assigns or updates the vehicle for the instructor for the driving lesson.

Lesson Class

Attributes:

* lessonID: int – A unique identifier for each lesson.
* studentID: int – The identifier for the student taking the lesson
* instructorID: int – The identifier for the instructor conducting the lesson
* vehicleID: String – The identifier of the vehicle assigned to the lesson
* date: Date – the date when the lesson is scheduled
* startTIme: String – The start time of the lesson
* endTime: String – the end time of the lesson
* status: String – the current status of the lesson (Scheduled, completed, canceled)
* location: String – The pick-up and drop-off location for the driving lesson.
* instructorFeedback: String – feedback left by the instructor regarding the lesson.

Methods:

* scheduleLesson(studentID, instructorID, date, startTime, location): void – Schedules a new lesson through the student, checks if student has susceptible hours.
* modifyLesson(lessonID, newDate, newStartTIme): void – Allows the student or admin to modify the time and date of lesson that’s already scheduled.
* cancelLesson(lessonID): void - cancels the scheduled lesson.
* trackLessonProgress(lessonID): void - tracks the progress or completion status of a lesson.
* notifyInstructor(instructorID): void - Sends a notification to the instructor when a lesson is scheduled or modified.
* recordInstructorFeedback(lessonID, comments): void – Allows the instructor to provide feedback after the completion of a driving lesson.

Package Class

Attributes:

* packageID: int – Unique identifier for each package
* packageName: String – Name of the package (Package One, two, or three)
* packageDescription: String – Brief description of what the package offers, and what resources it gives
* packageHours: int – Total hours included in the package.
* price: double – Cost of the package
* available: Boolean – indicates if the package is currently available for purchase or not.

Methods:

* AddPackage(): - Adds a new driving package to the system
* RemovePackage(): - Removes a package from the system
* UpdatePackageDetails(PackageID, newDetails): - Allows modification of package details such as name, hours, or price.
* CheckAvailability(PackageID): - Checks if a package is available for booking
* AssignSession(StudentID, InstructorID): - Assigns a driving session to a student based on their package and the instructor’s availability.
* CalculateTotalPrice(PackageID): - Calculates the total price based on the selected package and student’s previous purchases
* ListAvailablePackages(): - Lists all available packages for students to choose from.
* TrackPackageUsage(PackageID, StudentID): - Tracks the number of hours/sessions a student has used from the package.

Practice Tests Class

Attributes:

* testID: int – Unique identifier for each practice test, since they are updated through the DMV
* testName: String – Name for easier categorization of each test.
* date: Date – Date of the DMV’s update for the test.
* testQuestions: List<String> - A list of questions included in the test.
* packageID: int – The ID of the driving package the student currently has
* elegibility: boolean – student eligibility depending on their package
* testAnswers: List<String> - The corresponding correct answers for the questions.
* studentID: int – The ID of the student taking the test.
* score: int – The student’s score on the test.
* feedback: String – Feedback provided after the student completes the test.

Methods:

* startTeest(studentID): void – Initiates the test for the student.
* submitTest(studentID, answers): void – Submits the test and calculates the score based on the student’s answers.
* reviewTest(testID): void – Allows the student to review their completed test and feedback.
* retryTest(restID): void – Allows the student to retake the test.
* updateTest(testID): void – the DMV updates these tests on a weekly basis. This needs to run on a clock automatically.

Online Lecture Class

Attributes:

* classID: int – Unique identifier for each online class
* className: String – Name of the online class (DMV Rules Overview)
* classContent: String – URL or path to the video tutorial or online material
* duration: int – The duration of the class in minutes.
* packageID: int – The ID of the driving package, and the level of authorization for the students
* classAuthorization: String – depending on the packages available, there are specific benefits provided to the student depending on what level of packages they own.
* classStatus: String – The status of the class (Active, Archived)
* instructorID: int – The instructor responsible for creating or managing the class.
* completionStatus: boolean – Tracks whether the student has completed the class.

Methods:

* startClass(studentID): void – Initiates the class for the student and starts tracking progress.
* completeClass(studentID): void – Marks the class as completed for the student.
* trackProgress(studentID): void – Tracks the student’s progress through the class, such as time spent.
* provideFeedback(classID, feedback): void – Allows the student to give feedback on the class.
* viewClassmaterials(classID): List<String> - Provides access to any supplemental materials associated with the class.
* assignClassToPackage(classID, packageID): void – Links the class to a specific package.
* updateClassContent(classID, newContent): void – Allows the instructor to update the content of the online class.
* archiveClass(classID): void – Archives the class, marking it unavailable to new students.

System (DriverPass) Class

Attributes:

* systemVersionID: int – A unique identifier for the version of the system
* systemStatus: String – The current status of the system (Active, Maintenance, Offline)
* cloudProvider: String – The cloud provider used for the system (AWS or Azure)
* databaseType: String – The type of database used (MySQL or PostgreSQL)
* securityLevel: String – The level of security used for encryption and authentication.

Methods:

* syncWithDMV(): void – Syncs the system with DMV to update the latest rules, test questions, and regulations
* generateActivityReport(): void – Generates a report detailing the system usage, such as reservations for lessons, cancellations, purchasing of packages, which user IDs are associated with these, and the system versions and listings of changes.
* backupData(): - Creates a backup of all system data on a daily basis.
* restoreData(): - Restores data from a previous backup in case of failure or data corruption.
* handleUserLogin(userID, password): - Authenticates a user trying to log into the system.
* monitorPerformance(): - Tracks the performance of the system, including load times, errors, uptime, and user feedback.
* updatePackages(): - Updates the driving lesson packages, including adding or disabling specific packages.
* resetPassword(userID): - Allows admins to reset passwords for users who have forgotten them.
* validateTransaction(transactionID): - verifies if a payment or transaction is valid and completed successfully.

EmailService Class

Attributes

* serviceProvider: String – The email service provider being used (Google, Yahoo, Outlook)
* emailAddress: String – The email address from which the system will send emails.
* isVerified: boolean – whether the email address has been verified.
* apiKey: String – The API key used for secure email transmission.
* twoFactorProvider: String – if being verified through a two factor authenticator, the email will hook up with the twoFactor’s API system, and generate updates through the app.

Methods

* sendVerificationEmail(userEmail): void – Sends a verification email to the user’s email address for account activiation.
* sendPasswordResetEmail(userEmail): void – Sends a password reset link to the user’s email address when requested.
* sendAppointmentNotification(studentID, instructorID): void – Sends a notification email to both the student and the instructor when a driving lesson is scheduled or modified.
* validateEmail(): Boolean – Verifies the email address from the email link to ensure it is valid.
* sendCustomEmail(emailAddress, subject, body): void – Allows the admin to send custom emails from the system.
* checkDeliveryStatus(emailID): void – Checks whether the email was successfully delivered or failed.
* logEmailActivity(emailID, status): void – Logs all email-related activities, such as sent status, to monitor success or failure.

Database (Cloud Infrastructure) Class

Attributes:

* databaseType: String - specifies the type of database being used (MySQL, PostgreSQL)
* cloudProvider: String – The cloud provider hosting the database (AWS, Azure, Google Cloud)
* dbConnectionStatus: boolean – Indicated whether the database is connected and operational.
* backups: List – System back ups are stored within the cloud, previous version are recorded on a weekly bases and used for disaster recoveries.
* storageCapacity: int – The amount of storage available for the database.
* encryptionStatus: boolean – indicates whether the data is encrypted.

Methods:

* storeStudentData(studentID, data): void – Stores the student-related information in the cloud database.
* retrieveStudentData(studentID): void – Retrieves the student’s data based on their unique ID.
* storeInstructorData(instructorID, data): void – Stores instructor-related information.
* retrieveInstructorData(instructorID): void - retrieves instructor data by their unique ID.
* storeLessonData(lessonID, data): void – stores details of a scheduled driving lesson.
* retrieveLessonData(lessonID): void – retrieves data for a specific driving lesson.
* createBackup(): void – Triggers a manual backup of the database.
* restoreBackup(backupID): void – Restores data from a specific backup in case of data loss or corruption.
* monitorPerformance(): void – Monitors the performance of the database in terms of load times, data integrity, and connection reliability.
* encryptData(): void – Encrypts sensitive information before storing it in the cloud database.

Relationships:

Admin 1..\* -> 0..1 Student: The Admin manages Students in the system by resetting passwords if requested. They can also remove a student if their account ends up showing up suspicious in the activity reports. The admin does not create or control student accounts directly. The relationship is a simple association because the admin performs management tasks but does not have exclusive control over Students. There can be multiple students managed by one admin, thus it’s a 1..\* -> 0..1 relationship.

Admin 1.. -> 0..\* Package: The Admin manages Packages by creating, modifying, or removing them as needed. The Admins can update all details of the packages, including the price, total hours of lessons, and benefits such as downloadable notes, practice tests, or lectures. Admins are responsible for ensuring that the packages align with business needs and market demands. They can disable packages if they desire, edit current ones, or add new ones. While packages are purchased by students to access DriverPass’s services, only admins have control over the availability and attributes of these packages. The admin can manage multiple packages, and multiple packages can exist within the system but require admin intervention for any changes or updates.

Admin 1..\* -> 1 System: The admin interacts with the Database indirectly through the System to monitor system activity, retrieve reports, and track user behavior such as payments, purchases, transactions and lesson scheduling. The admin relies on the system ability to pull data from the database, including record of system changes, user activity logs, and system changes during maintenance. This data helps the admin detect any suspicious behavior, such as irregular transactions or unauthorized user action. While the admin doesn’t directly manage the system infrastructure, they have access to reports generated by the system to maintain oversight. The system serves as the source of all the data related to user activity and system operations, which the admins accesses to maintain security and oversee transactions.

Student 0.. -> 1..\* Package: A student can purchase 1..\* packages, or multiple packages over time, but only one can be active. The student selects and purchases a Package, which grants them access to various DriverPass features. These packages provide different levels of benefits, such as access to online video lectures, DMV rule notes, and varying hours for in-person driving lessons. The availability of these packages is dependent on what the admin has configured in the system. Once a package is purchased, it defines what the student can access within the system and how many hours they are allowed for driving lessons.

Student 1 -> 0..\* Lesson. A student can book multiple lessons, and their hours range based on their package level. They schedule the lesson through the DriverPass system, when booking a lesson, the student can choose the date, time, vehicle, instructor, and location. The number of lesson hours booked cannot exceed the maximum number of hours allotted by the package. Each lesson session is typically two hours long, and students can modify or cancel their lessons, provided they do not exceed their remaining hours.

Student 1..\* -> 0..\* Practice test: Students can take practice tests to prepare for the DMV exam. The system grades the tests and provides feedback on incorrect answers, helping students improve their chances for the real tests. Students can have multiple attempts on practice tests, and multiple students can access the same practice tests. Their ability to take said practice tests depends on what package level they currently possess. If they have the authorization, they can download a separate executable to run the test. The practice tests are kept up to date through a RESTful API with the DMV’s latest rules and regulations, as they change on a weekly basis. Currently only students with a level 3 package can access the practice test option.

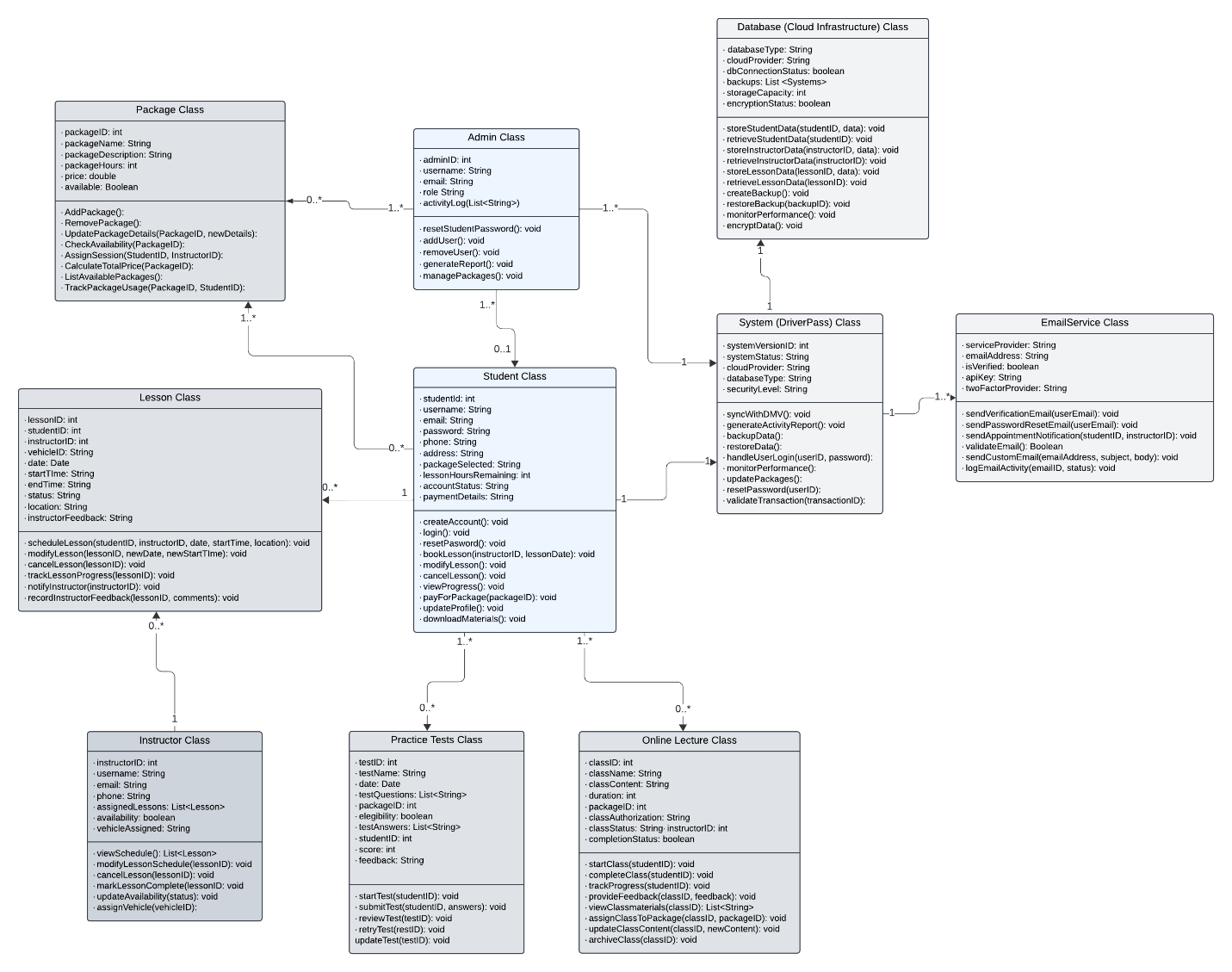
Student 1..\* -> 0..\* Lecture: multiple students can access the same lecture, and a student may attend multiple lectures depending on their package. These lectures include video content and downloadable notes on driving techniques, DMV rules, and safety protocols. The student can access and review these materials as part of their learning process. The ability to attend lectures or download notes is determined by the package level they have chosen, with higher-tier packages offering more comprehensive access to online learning materials.

Student 1 -> 1 System: The system supports all of the student’s activities. It serves as the central platform for the student’s experience, offering access to all the necessary features such as lessons, tests, scheduling, account management, and purchasing packages. The system tracks the student’s login credentials, purchase history, lessons taken, and tests results. Additionally, it manages their personal information such as their account details and identity verification. The system provides a user interface for students to navigate all the features mentions. The system also supports security features such as login validation, password resets, encryption services of sensitive information through the cloud architecture. It also provides security for itself, tracking all of the student’s actions, providing records of their activities such as scheduled or canceled lessons, purchases, and test performance. The system operates as the core environment that ties all the student’s interactions.

Instructor 1 -> 0..\* Lesson: An instructor may have multiple lessons scheduled but no lesson can exist without an instructor. The interview doesn’t directly cover all the functionalities of the instructor; however, it would be ideal that the instructor should be notified when a student requests a lesson based on the instructor’s availability. Instructors should have the flexibility to confirm or cancel the lesson, as well as communicate with students regarding scheduling or additional instructions. The system ensures that lessons are scheduled within the instructor’s availability and provides real-time updates to both parties.

System 1 -> 1 Database: The system relies on the Database (hosted in the cloud) to store and retrieve critical information such as system versions, student data, payment records, lesson schedules, and user activity. The database acts as a highly flexible and large storage unit, ensuring that data is secure through encryption methods such as SSL/TLS and is available for retrieval whenever needed. It also provides disaster recovery capabilities in the event of system corruption or failure, allowing for system rollback to previous stable versions. The cloud infrastructure also handles hardware-related issues, such as scaling and backup operations, removing the burden from DriverPass’s hardware limitations.

System 1 -> 1..\* Email Service: The system can be configured to work with multiple email providers, for essential functions such as email verification, password recovery, and notifications related to user activity. When a student or admin registers a new account, schedules a lesson, or requests a password reset, the system uses the email service to send out verification links or notifications. This ensures that all important communications, such as account creation, lesson confirmations, and security alerts, are handled securely and efficiently through the email provider. Emails also play a critical role in two-factor authentication and notifying students or admins of any updates, changes, or security concerns.



## Technical Requirements

*[Based on the diagrams you have created, describe the technical requirements of your system. These requirements should address the required hardware, software, tools, and infrastructure necessary for your system design.]*

**Hardware Requirements:**

* **Cloud-Base Infrastructure:** The system will have to be hosted on cloud platforms such as AWS or AZURE. this allows for a minimization of technical problems allowing DriverPass to focus on in house systems without having to worry about hardware related issues. Several of the benefits included with the cloud are that it allows scalabilities with the number of devices connected to the server, meaning that if more users enter the server, it will not affect performance.
* **Devices:** All users (students, admins, and instructors) will have to connect to the internet through a device. This includes desktops, laptops, tablets, and mobile phones. These devices are commonly integrated with modern browsers like Chrome, Safari, or Edge, which is what the web-based system is designed to run on. The IT Officer and Admin will likely have to have access to their own set up, along with a public computer or computer room for students.
* **Network Equipment:** Since this is hosted through the web, routers used for a network infrastructure for the office will be paramount to ensure a publicly available internet connection open to all. Certain brands like NETGEAR, ASUS, or TP-LINK can offer high quality WIFI across the building.

**Software Requirements:**

* **Operating Systems:** The system will be web based, meaning it can run on modern browsers such as Chrome, Safari, and Edge. These have the added benefit of being compatible with all of the modern operating systems such as Windows, Linux, macOS, android, and iOS. This ensures compatibility and flexibility for users regardless of the device they currently possess.
  + **Scalability:** the system must be designed with scalability in mind. The system must support mobile devices with a scaled down UI to accommodate the format of mobile devices. Things like larger buttons to work with a touch screen navigation, smaller resolutions, and power conservative methods for the weaker hardware on said mobile devices. Statistics show that over 50% of technology-based users are actually on mobile devices, and it was mentioned during the interview that the Instructors and Students may need DriverPass on the go. It will be quite the investment as tools like React Native, and Flutter can be integrated to convert web-based designed systems into mobile apps. Many other considerations such as UI scaling, API reintegration, testing tools,.
    - **UI Scaling:** This can be done with Material UI or Onsen UI, which includes tools for building hybrid apps with mobile-looking UIs for iOS and Android.
    - **API reintegration:** AWS Amplify or Azure will have to reintegrate the different hardware between the client and the server across different platforms.
    - **Optimizations:** This can only be done through the front-end development, whether DriverPass was made through HTML5, or JavaScript. Methods such as Webpackking, lazy loading, code splitting, or image compression and image downsizing will have to be done.
* **Database Management Systems:** The system will rely on a cloud-based relational database such as MySQL or PostgreSQL to store user data, lesson history, payment information, and practice test results. These databases will regularly handle backups and disaster recovery protocols, along with encryption methods to help protect against breaches.
* **Application Framework:** The main framework that’ll be used to design DriverPass and put it on modern browsers would be from either of the popular client-side browser languages. HTML5, CSS3, and JavaScript, this will allow for the system to be compatible with modern browsers like Chrome, Safari, and Edge. As for the server-side backend development. Node.js, Python, and PHP will be good handling the transfer of databases, authentication, transactions, etc. The backend will include secure session management and RESTful APIs for interaction with external systems like the DMV or email providers (Google or Outlook).
* **Security:** The use of HTTPS and **SSL/TLS** for encrypted communication between the client and the server will also be implemented to allow for private transfers of data thus stopping any third parties from intercepting or tampering with said information. This allows for sensitive information like Login credentials, test scores, payment information to be kept secure and away from unwanted eyes.
  + **Email Verifications & Two-Factor Authenticators**: To prevent brute force attacks, extra layers of security on top of a password may be enforced to keep accounts secure. Students may have to provide an email verification or a two-factor authenticator if an unknown IP address or an unusual amount of attempt is detected to attempt entering an account.

**Other Tools:**

* **Video Learning Tools:** AWS elemental media or connecting an API to Youtube can be used for hosting and delivering video tutorials for driving lessons.
* **Version Control:** the system will be deployed on a cloud platform (tools) like **AWS or Azure**, which will handle other benefits such as scaling, security, and availability of the services. It will also provide disaster recovery. Other tools like **Git** may be used for tracking specific changes that the Admins might want to locate when creating dashboards and charts. This will allow them to handle multiple versions and roll back in case of failures.
* **Dashboard Tools:** Like chart.js or D3.js can be used to provide visual data that the admins are requiring to track things like the number of transactions, appointments scheduled, and system changes.
* **Testing Runs:** Automated testing tools for web applications ensure compatibility across different browsers such as Chrome, Safari, Edge, and Mobile. There are tools like Selenium or Cypress to help see if the system is functional.
  + **Postman:** is another tool useful for testing APIs, ensuring the integrations between the DMV and email service third-party systems are working seamlessly.
* **Error Reporting:** Sentry is a tool that can provide real-time error reporting and logging to track issues in the system and allow developers to diagnose and address problems quickly.
  + **System Performance:** On the load times and database retrieval side and network performance, Datadog, CloudWatch can be used to track system performance, server load, and database load times in real time.
* **Google Analytics:** This can be useful for tracking user engagement, such as for things like page visits and system interactions. This will help IT officers understand what are the places with high user traffic.
* **Email Integration:** Integrating email services like Google, Yahoo, and Outlook through an external API for verification, notifications, and password resets will be required for this system.

**Infrastructures:**

* **Cloud Infrastructure:** The Cloud Provider (AWS or Azure) will host the system, offering cloud-based infrastructure to mange the platform’s hardware limitations. Solving problems like scalability with user, flexibility, and disaster recovery while minimizing downtime. It will allow DriverPass to handle fluctuating user demand without performance degradation.
  + **Store Services:** AWS will also be used for storing large files like video tutorials, practice test data, user-trafficking, previous system versions, etc. These services will ensure that files are stored securely and are accessible from any location.
  + **Backup & Recovery:** Cloud infrastructure allows for automated backs up and disaster recovery solutions to minimize data loss and downtime in case of system failures.
    - **Backup & Disaster Recover:** Backup systems like AWS Backup or Azure Backup will ensure that all critical data (user profiles, test results, scheduling information) is regularly backed up and stored for the sake of preservations in DriverPass and reliability. On a daily bases previous version of DriverPass along with all of its data and infrastructures will be saved in the form of a backup. This will allow DriverPass to roll back the system to a previous stable state should any emergencies occur.
  + **Scalability & Flexibility:** Cloud services provide elasticity, meaning the system can automatically adjust its resource allocation based on the current demand. During high traffic periods, more resources will be allocated, while resources can be scaled back during quieter times.
* **Networking:** The system relies heavily on reliable internet access for customers, instructors and admins.
  + **Connection Services:** AWS Direct Connect or Azure ExpressRoute is available to ensure dedicated, secure connections to the cloud infrastructure.
  + **Firewall & Security Gateways:** A cloud-based firewall should be used to protect the system from external attacks such as malware or spyware, which could create breaches or unauthorized access, controlling traffic into and out of the cloud environment.
  + **Virtual Private Network (VPN):** If additional security is needed for admin level users and IT Officers in the event that there was an unknown breach, to investigate anonymously, it would help to integrate a into the admin privileges.
* **Database Management:** on top of being through the cloud, databases such as MySQL or PostgreSQL will be used for managing user data, scheduling information, lesson history, payment record, and DMV practice tests. They provide high availability with their short retrieval times and contracts that ensure nearly perfect reliability.
  + **Scalable:** Cloud-based databases automatically scale to handle the increasing load of new users, ensuring that performance is not impacted by user growth.
  + **Content Delivery:** A content delivery network like CloudFront or Azure CDN will ensure that static content such as videos, notes, and tutorials are provided through the web. CDNs ensure fast loading times for users regardless of their geographic location**.** The scalability benefit applies to this too, as more content is added to DriverPass the size of the database also increases.
* **APIs & Integration:** The system must integrate with DMV systems to ensure up-to-date practice tests and rules are available to users. This requires secure, real-time API communication with the DMV for retrieving and updating data.
  + **Email & Messaging API:** Services like AWS SES, Twilio, or SendGrid will be necessary for sending notifications to users regarding appointments, updates, and lesson reminders.
* **Security and Compliance:** Security Groups & Role-Based Access should be implemented across the system. There should be strict access control policies using cloud-native tools like AWS IAM or Azure Active Directory. These tools enforce permissions based on user roles (admin, student, instructor).
  + **Encryption:** tools are provided automatically through cloud services such as AWS or Azure. Being transitted through SSL/TLS for secure communications.
* **Monitoring & Maintenance:** The IT Officer would greatly appreciate tools like Amazon CloudWatch, Azure Monitor or third-party systems like Datadog or New Relic. These systems will be used to monitor system performance and database activity
  + **Logging in & Error Tracking:** Centralized logging services such as AWS CloudTrail, Azure Monitor Logs, or third-party tools like Sentry will be necessary to log errors, track user activity and identify security breaches.
* **User Access & Mobile Integration:** The system will run on modern browsers like Chrome, Safari, edge, and must be designed to ensure full compatibility with these platforms, along with mobile friend infrastructures. The system should be optimized for mobile access with a responsive design to accommodate smartphones and tablets. Tools like Bootstrap or Foundation can be used for mobile conversations.
* **Development Environment**: Development and deployment will be managed using tools like GitHub or Bitbucket. Version control can be done with Jenkins, CircleCI, or AWS CodePipline for continuous integration/continuous deployment. These tools help automate code testing, building, and deployment, ensuring smooth system updates.
  + **Infrastructure of Code:** Automation tools like AWS CloudFormation or Azure Resource Manage will automate the setup of cloud resources (servers, databases, load balances) using infrastructure as Code principles. This ensure that infrastructure can be consistently deployed and managed across environments.

References:

<https://www.geeksforgeeks.org/use-case-diagram/>

<https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/>