# CS 255 System Design Document

Tishana Clayton

## UML Diagrams

### UML Use Case Diagram

### 

**The UML use case diagram for the DriverPass system identifies the main actors and their interactions with the system’s functionality:**

**Actors:**

* **Customer: Schedules driving lessons, takes practice quizzes, and manages their accounts.**
* **Admin: Manages users, generates reports, and updates system settings.**
* **Secretary: Assists customers by managing lesson appointments and progress tracking.**
* **Driver: Provides driving instruction for scheduled lessons.**
* **DMV System: Interfaces with the DriverPass system for verification purposes.**

**Key Use Cases:**

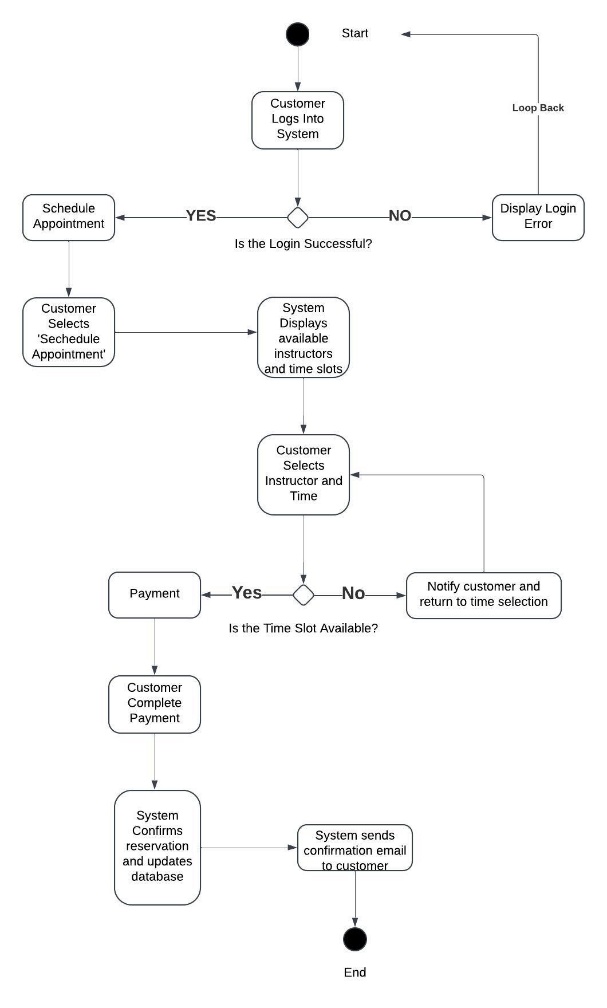
* **Customer:**
  + **Create Account**
  + **Login (includes Verify Login and extends Display Login Error)**
  + **Schedule Driving Appointment**
  + **Purchase Packages**
  + **View Study Materials and Take Practice Quizzes**
* **Admin:**
  + **Manage Users**
  + **Generate Reports**
  + **Update DMV Policies, Rules, and Questions**
* **Secretary:**
  + **Modify Appointments**
  + **Update Student Progress**

**Relationships:**

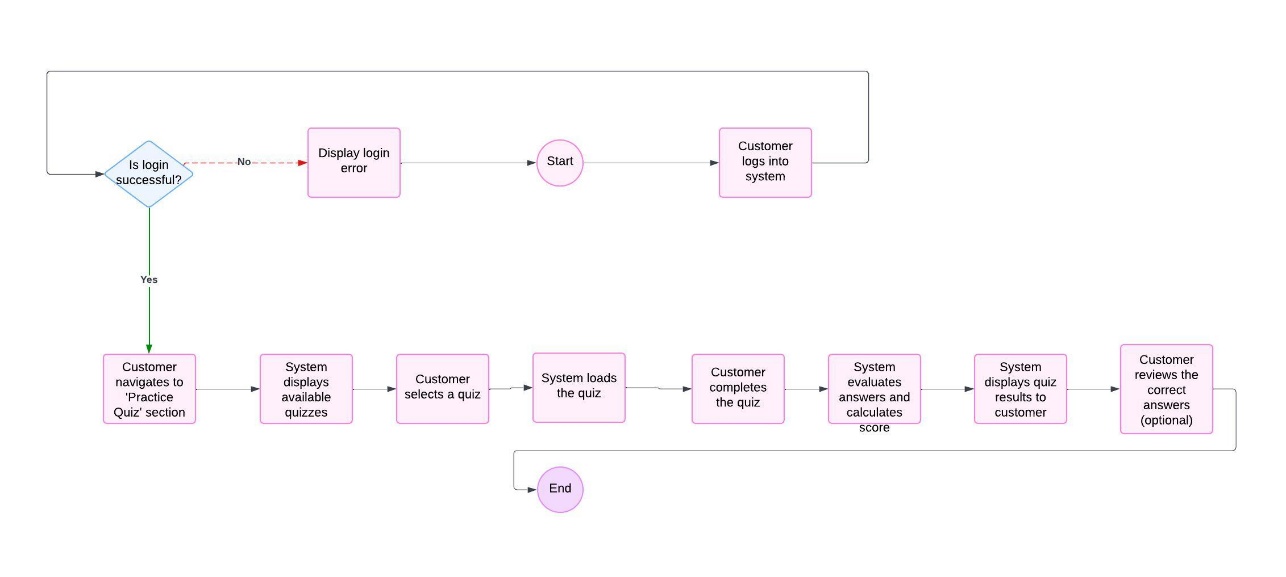
* **Use cases include mandatory actions where necessary (e.g., “Schedule Driving Appointment” includes “Verify Payment”).**
* **Optional or conditional actions (e.g., “Display Login Error”) extend their primary use cases.**

### UML Activity Diagrams

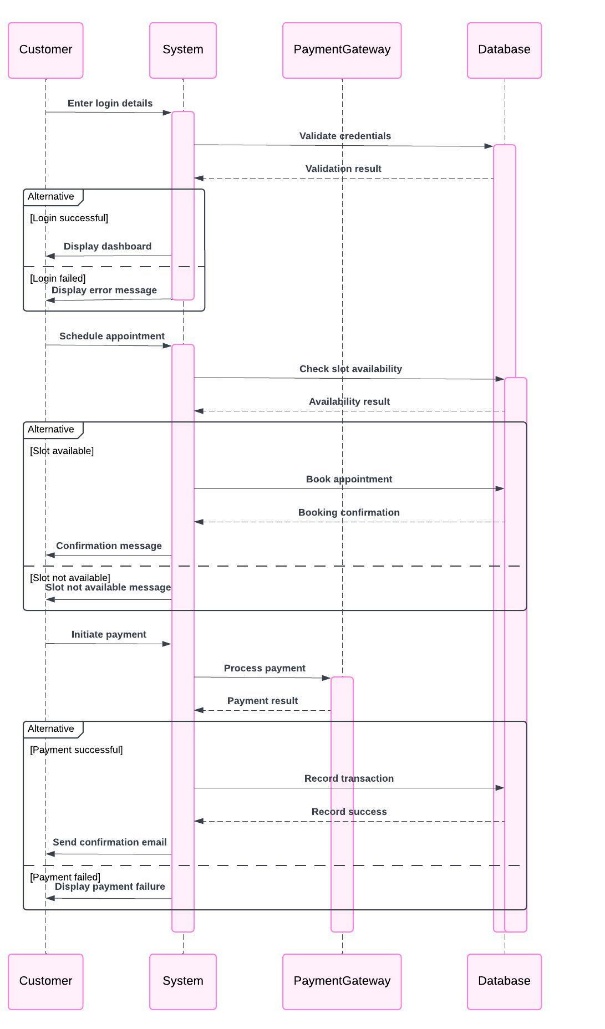
Scheduling a Driving Appointment

**

Taking a Practice Quiz

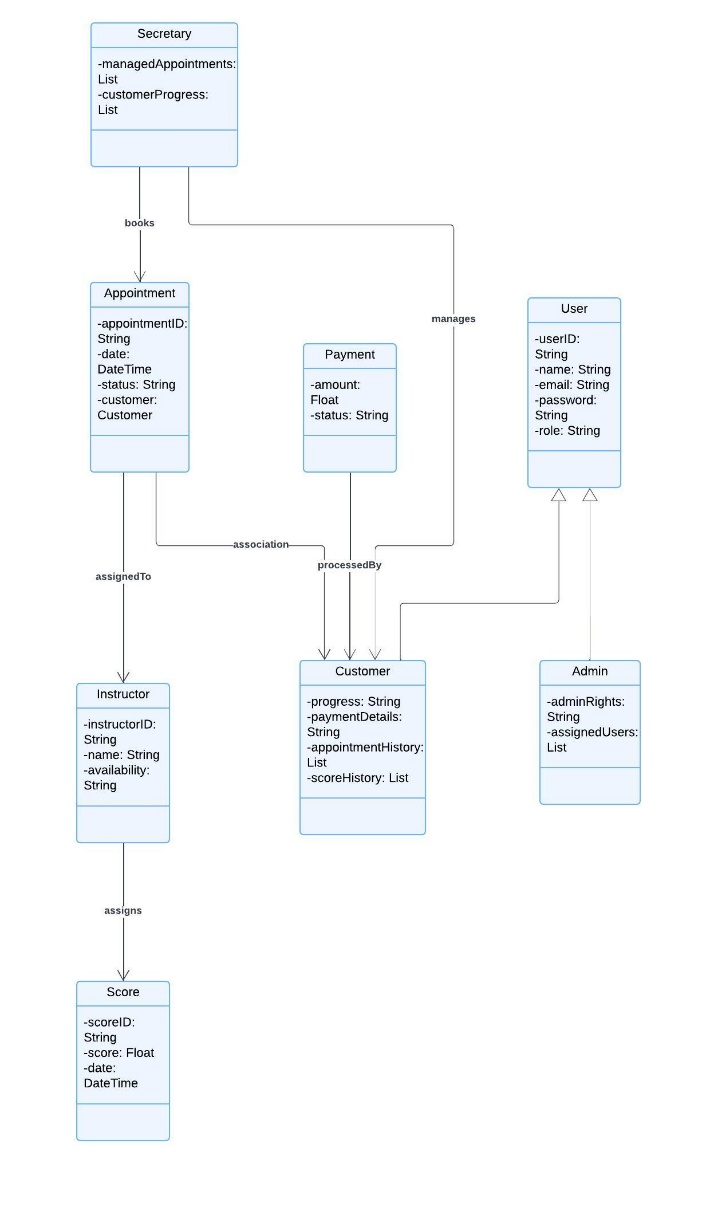


### UML Sequence Diagram

Scheduling A Driving Appointment  


### UML Class Diagram

DriverPass System UML Class Design

****

**The DriverPass system involves several classes, each representing key components of the system. The UML Class Diagram outlines the classes, their attributes, and relationships.**

* **User (abstract class or parent class for Customer, Admin, Secretary):**
  + **userID: String**
  + **name: String**
  + **email: String**
  + **password: String**
  + **role: String (Customer, Admin, Secretary, Instructor)**
* **Customer (inherits from User):**
  + **progress: String**
  + **paymentDetails: String**
  + **appointmentHistory: List<Appointment>**
  + **scoreHistory: List<Score>**
* **Admin (inherits from User):**
  + **adminRights: String**
  + **assignedUsers: List<User>**
* **Secretary (inherits from User):**
  + **managedAppointments: List<Appointment>**
  + **customerProgress: List<ProgressReport>**
* **Instructor:**
  + **instructorID: String**
  + **name: String**
  + **availability: String (Available, Not Available)**
* **Appointment:**
  + **appointmentID: String**
  + **date: DateTime**
  + **status: String (Scheduled, Completed, Canceled)**
  + **customer: Customer**
  + **instructor: Instructor**
* **Payment:**
  + **amount: Float**
  + **status: String (Pending, Completed)**
  + **customer: Customer**
* **Score:**
  + **scoreID: String**
  + **score: Float**
  + **date: DateTime**
  + **customer: Customer**
* **ProgressReport:**
  + **reportID: String**
  + **date: DateTime**
  + **progress: String (Passed, Failed)**
  + **customer: Customer**

## Technical Requirements

**1. Hardware Requirements**

**Servers:**

* **Cloud-based Servers**:
  + **AWS EC2** or **Azure Virtual Machines** for hosting the system. The system needs to be scalable to handle varying traffic, particularly during peak times (e.g., customer registration and payment processing).
  + **Initial Server Specifications**:
    - CPU: 2-4 cores
    - RAM: 8 GB
    - Storage: 100 GB SSD
    - Load balancing through cloud services to ensure continuous uptime and optimal performance.

**User Devices:**

* **Desktop**:
  + Users will access the system through standard web browsers (e.g., Chrome, Firefox, Safari) on Windows, macOS, and Linux devices.
* **Mobile Devices**:
  + **iOS** and **Android** devices for mobile users. The system must be responsive to support both desktop and mobile platforms efficiently.

**Backup Storage:**

* **Cloud Storage**:
  + **AWS S3** or **Azure Blob Storage** to store backups of critical data, including user accounts, appointments, and payment records.

**2. Software Requirements**

**Operating Systems:**

* **Web Servers:**
  + **Use Linux-based systems (Ubuntu, CentOS) or Windows Server for hosting the web application and API backend.**
* **Client-side:**
  + **Compatible with modern browsers running on Windows, macOS, Linux, iOS, and Android.**

**Web Framework:**

* **Frontend:**
  + **React.js or Angular for building a responsive, interactive, and user-friendly interface.**
  + **Supports mobile-first design and ensures smooth user interaction for both desktop and mobile users.**
* **Backend:**
  + **Node.js with Express.js or Java Spring Boot to handle server-side logic, API requests, and data processing. These frameworks provide robust scalability and efficient performance.**

**Database Management:**

* **Relational Database:**
  + **MySQL or PostgreSQL to manage structured data such as customer details, appointments, payments, and progress reports. The database will handle transactional data securely and efficiently.**
  + **NoSQL Database (Optional):**
    - **MongoDB for handling unstructured data like session data or logs.**

**Payment Gateway:**

* **Stripe or Square for handling secure credit card transactions, including scheduling fees, and ensuring safe and seamless payments.**

**Authentication & Authorization:**

* **OAuth 2.0 or JWT (JSON Web Tokens) for handling secure authentication and role-based access (Admin, Secretary, Customer).**
  + **Two-Factor Authentication (2FA) for admins and secure operations.**

**3. Tools and Infrastructure**

**Cloud Hosting & Infrastructure:**

* **Cloud Platform:**
  + **AWS or Microsoft Azure to ensure the system can scale based on user demand and remain highly available. This also supports redundancy and disaster recovery.**
  + **Docker for containerization of the application, ensuring it can be deployed in any environment with minimal setup.**
  + **Kubernetes (for large scale) to manage containerized applications, ensuring the system is scalable and resilient.**

**Development Tools:**

* **Version Control:**
  + **GitHub or GitLab for collaborative version control and code management. Allows multiple developers to work on different parts of the system.**
* **IDEs:**
  + **Visual Studio Code or IntelliJ IDEA for efficient coding, debugging, and running the system.**

**API Documentation & Testing:**

* **Swagger or Postman for documenting and testing APIs to ensure that all backend services are properly defined and can be easily tested by developers.**

**Monitoring and Maintenance:**

* **AWS CloudWatch or Datadog for monitoring the health of the application and its components.**
* **Log Management:**
  + **ELK Stack (Elasticsearch, Logstash, Kibana) for managing logs and visualizing system activities.**

**Automated Testing:**

* **Jest (for JavaScript testing) or JUnit (for Java-based systems) for automated unit and integration testing.**
* **Selenium or Cypress for frontend UI testing to ensure the system works as expected across different browsers and devices.**

**Continuous Integration and Deployment (CI/CD):**

* **GitLab CI or Jenkins to automate the build, testing, and deployment process. Ensures that code changes are thoroughly tested before being deployed to production.**

**4. Security Infrastructure**

**SSL/TLS Encryption:**

* **HTTPS encryption for secure data transmission between clients and the server. All sensitive data (user credentials, payment details) will be encrypted during transit.**

**Firewalls:**

* **AWS Security Groups or Azure Network Security Groups to define and control traffic between the web server, database, and external services.**

**Backup Encryption:**

* **AES-256 encryption for sensitive backup data, ensuring that data stored in backups is protected.**

**Data Integrity:**

* **Use hashing algorithms (e.g., SHA-256) to securely store sensitive information like user passwords.**

**5. Scalability Considerations**

* **Auto-Scaling**:
  + **AWS Auto Scaling** or **Azure Autoscale** to automatically scale resources (e.g., servers, databases) based on demand, ensuring the system can handle fluctuating traffic.
* **Load Balancers**:
  + **AWS Elastic Load Balancer** or **Azure Load Balancer** to distribute traffic evenly across multiple servers to ensure system reliability and uptime.
* **Database Scalability**:
  + Use **Read Replicas** in **AWS RDS** or **Azure Database** to handle increased read traffic, allowing for better performance and availability.

**6. Network and Communication Requirements**

* **High-Speed Internet Connectivity**:
  + Ensure a fast, reliable internet connection for both the cloud infrastructure and end users.
* **Content Delivery Network (CDN)**:
  + **AWS CloudFront** or **Cloudflare** to cache static assets and improve the speed of content delivery to users globally.