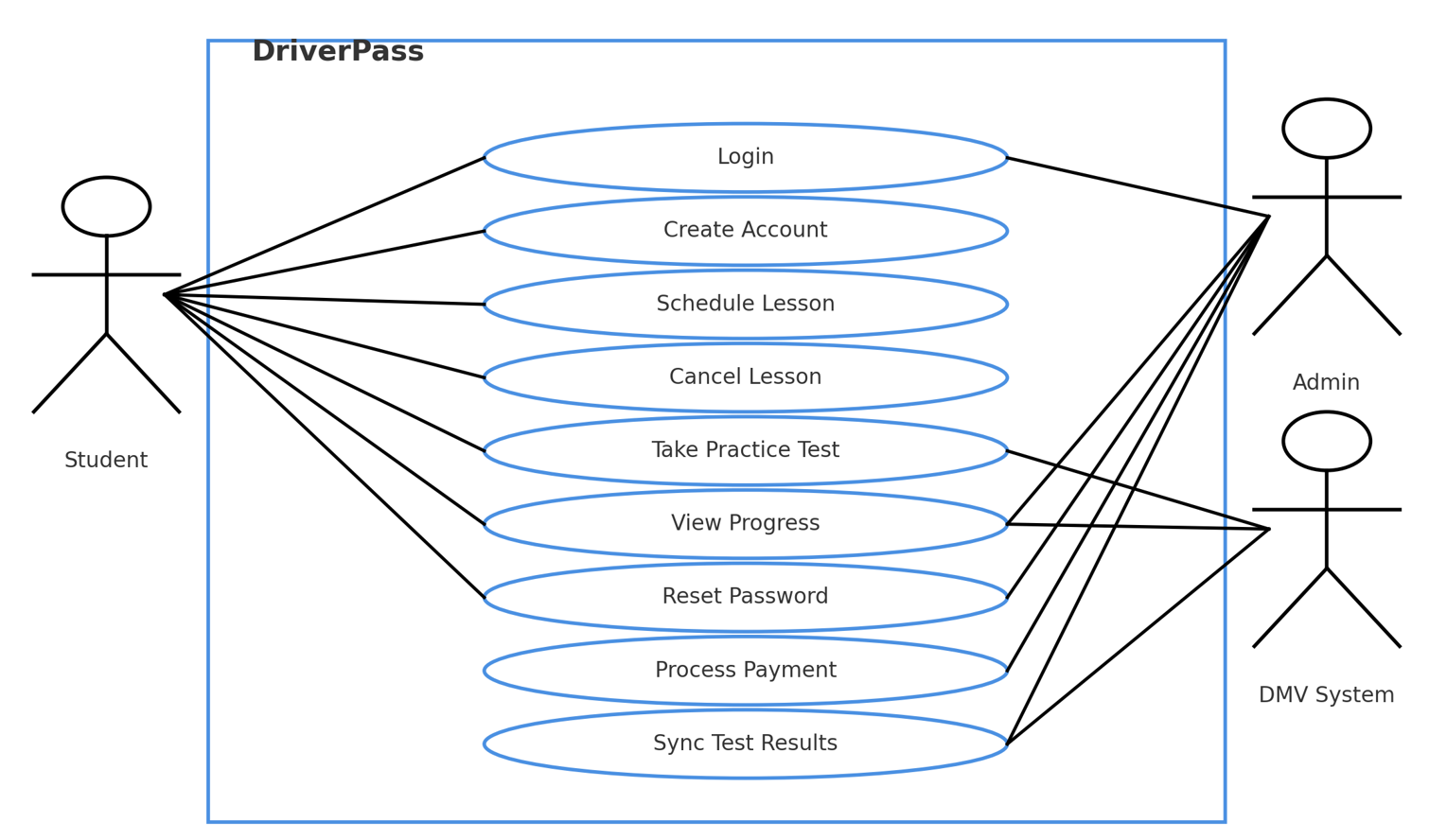
# CS 255 System Design Document Template

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

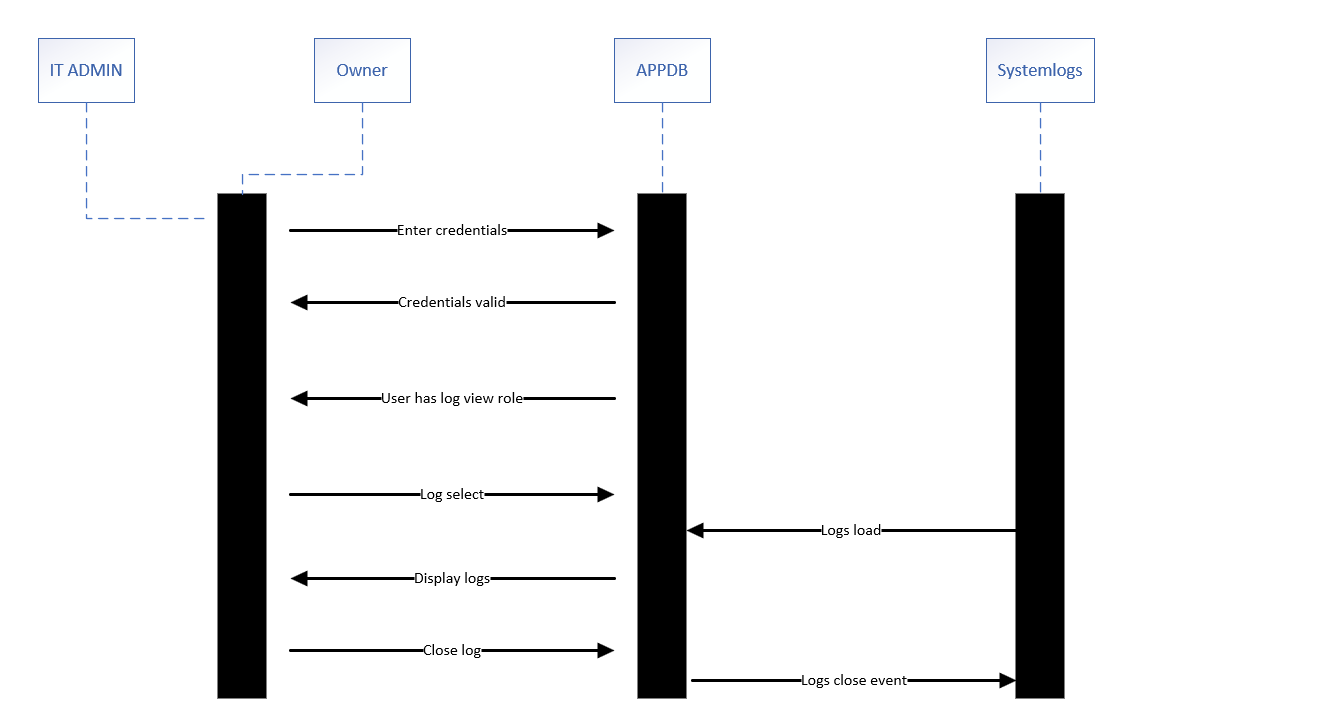


### UML Activity Diagrams #1 – Reset password

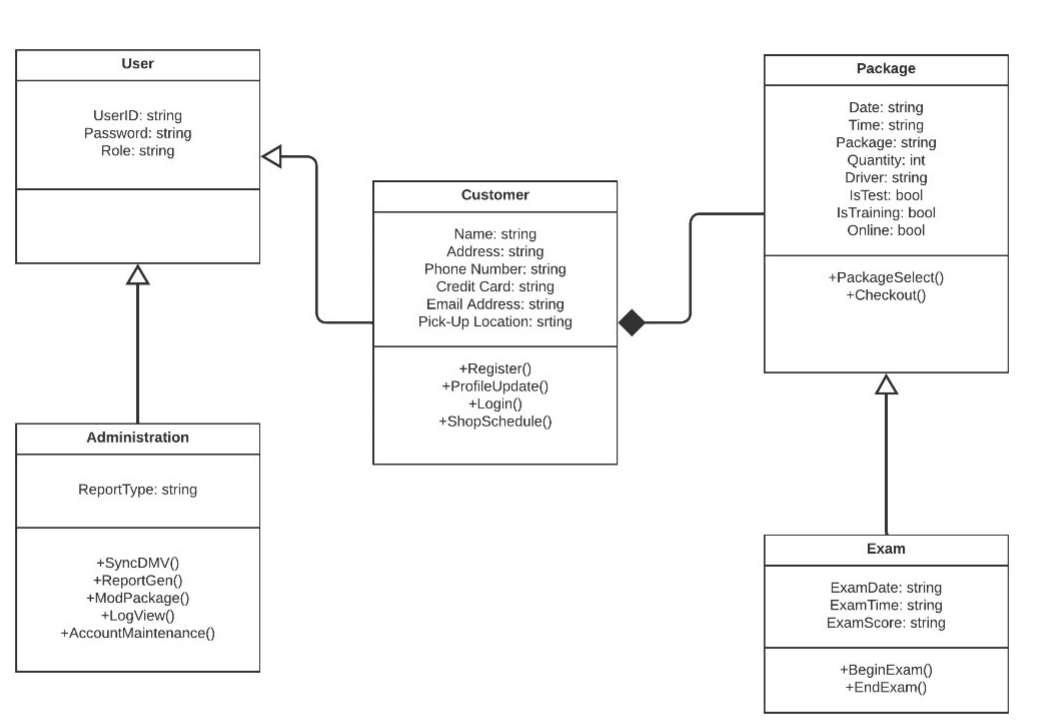
### UML Activity Diagrams #2 – Viewer log

### 

### UML Sequence Diagram

**

### UML Class Diagram

**

## Technical Requirements

1. Platform and Hosting (Infrastructure)

* Cloud-hosted web application on AWS, Azure, or GCP.
* Three tiers: web (HTTPS load balancer and CDN), application (stateless container or VM pool), and database (managed relational database).
* Separate environments: Development, Test/QA, Staging, Production.
* VPC/VNet with private subnets for app and data tiers; security groups/NSGs; NAT gateway for outbound calls (DMV).
* Auto-scaling for app tier; database read replicas or scale-up as needed.
* High availability with multi-AZ or multi-region load balancing; recovery objectives: RPO ≤ 15 minutes, RTO ≤ 4 hours.

1. Hardware (Capacity Targets)

* App nodes: 2–4 vCPU and 8–16 GB RAM per node with horizontal scaling.
* Database instance: 4–8 vCPU and 16–32 GB RAM with provisioned IOPS storage; daily full backups and 7–30 day point-in-time recovery.
* Storage: object storage bucket for exports and reports (start at 100 GB), encrypted volumes for logs and temp files.
* Managed load balancer and CDN.

1. Software Stack (Application)

* Frontend: responsive web UI built with HTML5 and a modern framework such as React, Vue, or Angular; WCAG AA compliant.
* Backend API: RESTful service using Node.js/Express, Python/FastAPI, Java/Spring Boot, or .NET 8.
* Database: managed PostgreSQL or MySQL with ORM and schema migrations.
* Payment processing: PCI-compliant gateway such as Stripe, Braintree, or Adyen with tokenization (no raw card storage).
* External integrations: DMV system via secure REST/SOAP or file transfer, signed requests, retry and dead-letter queue; Email/SMS provider (SES, SendGrid, Twilio).
* Background jobs: queue and worker for DMV sync, report generation, and notifications (e.g., SQS/Lambda, RabbitMQ, or Azure Service Bus).

1. Security Requirements

* Enforce HTTPS/TLS 1.2+ with HSTS and strong ciphers.
* Authentication with hashed passwords (bcrypt or Argon2) and optional MFA for admin users.
* Role-based access control covering Student/Customer and Admin/IT/Secretary roles.
* Use a cloud secrets manager for database credentials, API keys, and payment tokens.
* Data encryption at rest (AES-256) and in transit; minimize stored PII; mask sensitive data in logs.
* Apply OWASP Top 10 protections: CSRF tokens, input validation, parameterized SQL, output encoding, rate limiting, and content security policy.
* Maintain audit logs for admin actions; retain logs for 90–365 days.

1. Observability and Operations

* Centralized, structured logging with trace IDs.
* Application performance monitoring for latency, error rates, database performance, and queue depth.
* Alerting for SLA breaches, spikes in auth failures, payment errors, DMV sync failures, and resource saturation.
* Nightly full database backups with cross-region snapshot copies; tested restores.
* Compliance: handle training data similar to FERPA rules and use PCI DSS SAQ-A by tokenizing payments.

1. Developer Experience and Tooling

* Git-based version control.
* CI/CD pipelines that run tests and security scans, build container images, and deploy across environments with approvals.
* Docker containerization with vulnerability scanning and a private registry.
* Infrastructure as code with Terraform, Bicep, or CloudFormation.
* Testing strategy: unit, integration, API, UI (Playwright or Cypress), performance (k6 or JMeter), security scans (OWASP ZAP or Snyk), and contract tests for DMV integration.

1. Client Devices and Access

* Supported browsers: current and previous versions of Chrome, Edge, Firefox, and Safari.
* Mobile-friendly UI for students; desktop experience for admins with PDF viewing.
* Accessibility: keyboard navigation, ARIA roles, color contrast, and screen-reader labels.

1. Data and Domain Constraints

* Core entities: User, Customer, Package, Exam, Administration with normalized tables and foreign keys.
* Business rules: a Customer can select one or more Packages; Exam relates to a Package; unique indexes for username and email; constraints for package capacity and schedule conflicts.

Summary  
This system is a secure, scalable, cloud-hosted web application that supports the DriverPass use cases such as account creation, login, lesson scheduling and canceling, practice tests, payments, progress viewing, DMV synchronization, and reporting. The requirements above specify the hardware, software, tools, and infrastructure needed to move from design to a production deployment.