# CS 255 System Design Document

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

A diagram of a diagram

AI-generated content may be incorrect.

### UML Activity Diagrams

A diagram of a company

AI-generated content may be incorrect.

A screenshot of a diagram

AI-generated content may be incorrect.

### UML Sequence Diagram

A screenshot of a diagram

AI-generated content may be incorrect.

### UML Class Diagram

A screenshot of a computer

AI-generated content may be incorrect.

## Technical Requirements

**Architecture**

* Three-tier, cloud-hosted web application: Front End (Web UI) → API/Services → Relational Database.
* Stateless services behind a load balancer; all traffic over HTTPS (TLS 1.2+).
* Role-based access control (RBAC) enforced at the API layer; audit logging for all create/update/delete actions.

**Hosting & Hardware**

* Cloud provider: AWS/Azure/GCP (any is acceptable). Initial sizing:
  + API/Services: 2–3 containers/VMs, each 2–4 vCPU / 8–16 GB RAM.
  + Database: managed Postgres/MySQL instance (e.g., AWS RDS/Azure DB) with 2–4 vCPU / 8–16 GB RAM, GP SSD storage, automated backups.
  + Object storage for exports (CSV/Excel) and static assets.
* Horizontal scaling by increasing service replicas; vertical scaling by upgrading instance sizes.

**Software Stack**

* **Front End**: Responsive web app (HTML/CSS/JS). Framework allowed: React/Vue or plain MVC views.
* **Back End**: RESTful services (Java/Spring Boot or Python/FastAPI). Separate modules for:
  + **Scheduling Service** (availability, conflict checks, reservations)
  + **Testing Service** (practice tests, attempts, grading)
  + **User & Role Service** (accounts, roles/permissions, status/lock/disable)
  + **Notification Service** (email/SMS confirmations)
  + **Audit Service** (append-only activity log)
* **Database**: PostgreSQL (preferred) with normalized schema for Users, Roles/Permissions, Vehicles, Packages, Reservations, LessonNotes, PracticeTests, TestAttempts, AuditLog.
* Caching (optional, Phase 2): Redis for read-heavy endpoints (availability lookups).
* **External Integration**: optional DMV system lookup endpoint for permit validation (mock/stub for dev/test).

**Functional Alignment (from diagrams)**

* **Scheduling**: Prevent double-booking by atomic availability checks and transactions on Reservation create. Supports modify/cancel with status transitions.
* **Testing**: Randomized question pulls, attempt scoring, and status updates (Not Taken/In Progress/Failed/Passed).
* **Security Admin**: Create/disable users, assign roles, reset accounts, block access.
* **Reporting**: Exportable activity/audit reports; read-only CSV/Excel exports.

**Security Requirements**

* **Authentication**: Username/password with salted hashing (Argon2/bcrypt). Optional MFA.
* **Authorization**: RBAC mapped to API scopes; least-privilege by default.
* **Transport Security**: HTTPS only; HSTS enabled.
* **Data Protection**: Encryption at rest for DB and object storage; secrets stored in cloud secret manager.
* **Session Management**: Short-lived access tokens; idle/session timeouts; account lockout policies.
* **Auditability**: Who/what/when captured for all sensitive changes; immutable audit log.

**Availability, Backup & Recovery**

* **Backups**: Daily full + point-in-time recovery (PITR) window ≥ 7–14 days.
* **Uptime Target**: ≥ 99.5% (Phase 1). Health checks and rolling/blue-green deploys.
* **Disaster Recovery**: Automated restore runbook; RPO ≤ 24h, RTO ≤ 8h (initial targets).

**Observability**

* Centralized structured logging (request IDs, user IDs, error codes).
* Metrics & alerts: error rates, latency, booking failures, email/SMS bounce rates.
* Audit report generation success/failure tracking.

**DevOps & CI/CD**

* Git-based workflow, branch protections, code review required.
* CI: unit/integration tests, lint, SAST/dep scan.
* CD: staging → production with approvals; infrastructure as code (Terraform/Bicep).
* Environment parity: Dev, Test, Prod; separate credentials and networks.

**Data Management**

* **Schema**: As per class diagram; foreign keys and unique constraints (e.g., one vehicle/instructor cannot have overlapping reservations).
* **Migrations**: Versioned DB migrations (Liquibase/Flyway).
* **Seed Data**: Roles/permissions, default packages, example vehicles/instructors for testing.

**Notifications**

* Email provider (e.g., SES/SendGrid) and optional SMS provider (e.g., Twilio).
* Retry with backoff; bounce handling; templates for confirmations and reminders.

**Performance Targets (initial)**

* P95 API latency ≤ 500 ms for core reads; ≤ 1 s for booking/attempt submit.
* Booking concurrency: 50 parallel requests without conflict errors.
* Pagination on list endpoints; server-side filtering.

**Compliance & Privacy**

* Store minimum PII required (name, contact, permit number).
* Provide data export/delete on request (data subject request workflow).
* Access logs retained ≥ 90 days; audit logs ≥ 1 year (per policy).

**Assumptions & Constraints**

* Offline Excel usage is read-only; updates occur only through the web app.
* DMV integration availability may vary; system must operate if DMV is unreachable (graceful degradation).
* Initial user base is small (instructors/secretary/admin + students) with room to scale.