# CS 255 System Design Document Template

Selvadurai Pathmathasan

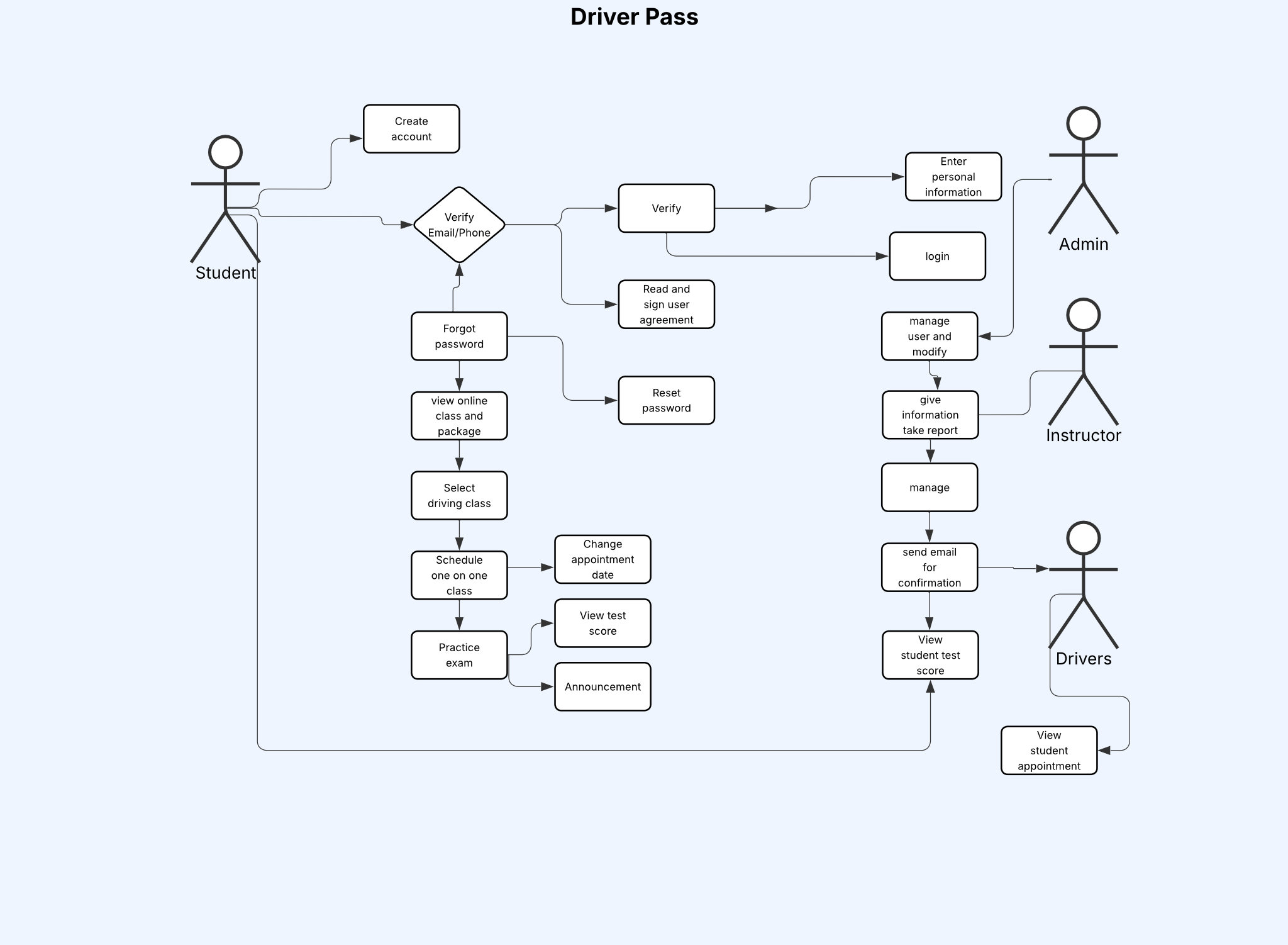
The Driver Pass System is a secure, role-based platform that manages driver registration, verification, and approval across four roles: admin, student, instructor, and driver. Drivers submit required documents during registration and track their application status via a personal dashboard. Admins oversee the entire process, verifying submissions, managing application statuses, and communicating with drivers through notifications.

The system enforces strong security with multi-factor authentication and limits login attempts to prevent unauthorized access. Instructors may support the approval process, while students have minimal access, primarily for viewing assigned drivers. Built using modern web technologies (Next.js, Rust/Go/Node.js, PostgreSQL), the system is containerized with Docker and deployed on Render for reliability, scalability, and ease of maintenance.

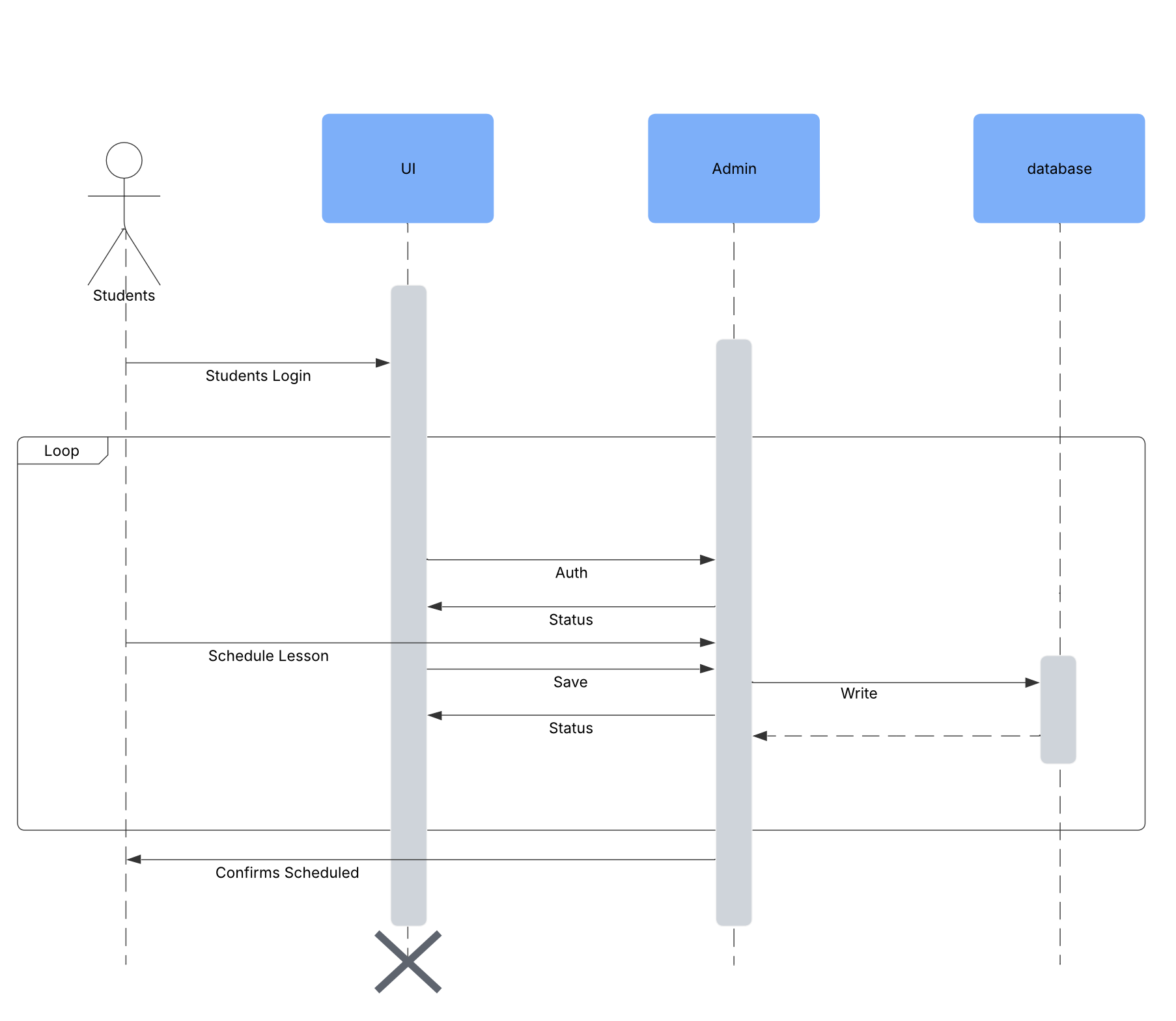
## UML Use Case Diagram

### Screenshot 2025-04-26 at 4.31.29 PM.png

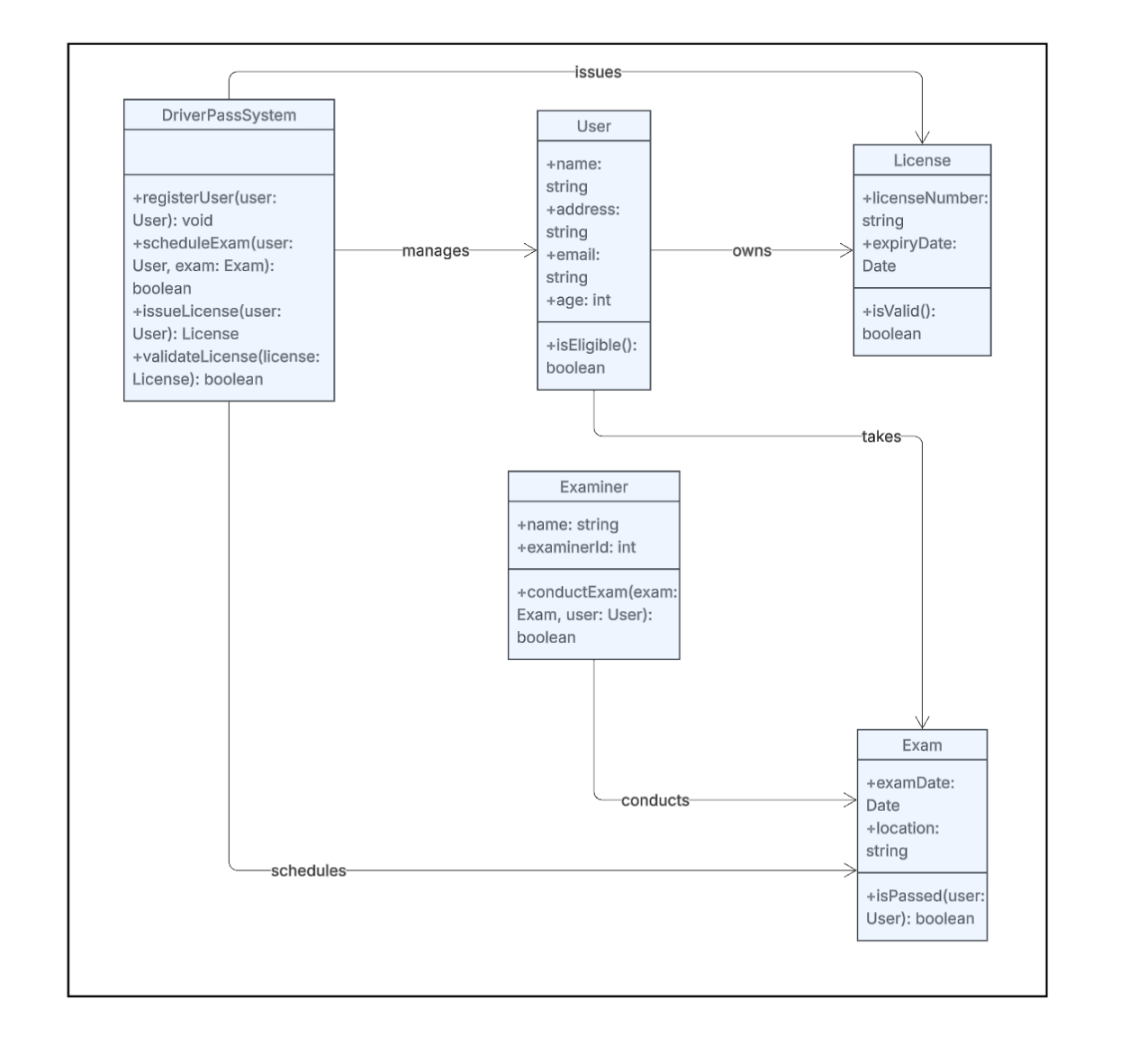
### UML Activity Diagrams



### UML Sequence Diagram

**

### UML Class Diagram



**Technical Requirements Document**

**1. Overview**

The Driver Pass System is a module designed to manage the registration, verification, and authorization of drivers. It integrates into an existing system with four defined user roles: **Admin**, **Student**, **Instructor**, and **Driver**.

**2. User Roles and Permissions**

**2.1 Admin**

* Full access to all modules.
* Can create, update, or remove users.
* Can approve or reject driver pass applications.
* Can manage system configurations and settings.

**2.2 Student**

* Can apply for services but does **not** interact with the driver pass system directly.
* Can view assigned driver details (if applicable).

**2.3 Instructor**

* Can view and recommend drivers.
* Can track student-driver pairings for lessons (if relevant).
* Read-only access to driver verification statuses.

**2.4 Driver**

* Can register and submit a driver pass application.
* Can upload required documents (license, ID, proof of insurance).
* Can view the status of their application (Pending, Approved, Rejected).
* Receives system notifications on status updates.

**3. Functional Requirements**

**3.1 Driver Registration**

* From capturing personal info: Name, Phone, Email, Address.
* Upload fields for license, insurance, and vehicle registration.
* Auto-validation of email and phone via OTP.

**3.2 Document Verification**

* Admin interface to view uploaded documents.
* Option to Approve or Reject with optional feedback.

**3.3 Application Workflow**

* Status states: Draft → Submitted → Under Review → Approved/Rejected.
* Rejection triggers email/SMS notification with reason.

**3.4 Dashboard**

* Admin Dashboard:
  + View total drivers, pending reviews, approved drivers.
* Driver Dashboard:
  + Application statuts
  + Edit profile
  + Resubmitted applications

**4. Technical Requirements**

**4.1 Backend**

* **Language**: Rust / Go / Node.js
* **Database**: PostgreSQL
* **Authentication**: JWT-based Auth
* **API Security**: Role-based access control (RBAC), Input sanitization

**4.2 Frontend**

* **Framework**: Next.js
* **State Management**: Redux Toolkit
* **UI Libraries**: Tailwind CSS, React Toastify for feedback

**4.3 Deployment**

* Containerized using Docker
* Hosted on **Render** or **Vercel** for frontend
* Auto-deploy via GitHub integration

**5. Security Considerations**

* Case-sensitive passwords with strong policy
* Multi-factor authentication (MFA) for Admins
* Account lockout after 3 failed login attempts
* Secure storage for uploaded documents (e.g., S3)

**6. Notifications**

* Email + in-app notifications for key actions:
  + Application submitted
  + Approved / Rejected
  + Document Expiry