High Level Design

on

Laravel k8s Automotive Management System

Course Name:

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1	YASHIT JAIN	EN22EL301069@medicaps.ac.in
2	YUGAL BISANI	EN22EL301070@medicaps.ac.in
3	YUVRAJ SINGH SIKARWAR	EN22CS3011129@medicaps.ac.in
4	YASHASVI NANNAWARE	EN22EL301068@medicaps.ac.in

Group Name: 03D10
Project Number: DO-16
Industry Mentor Name:
University Mentor Name: Prof Avnesh Joshi
Academic Year: 2025-2026

Table of Contents

1. Introduction

1.1 Scope of the Document

This is a High-Level Design (HLD) document for the Laravel K8s Automotive Management System. It serves as the architectural blueprint before development, explaining how the system is structured, what technologies are used, how it is deployed on Kubernetes, and the rationale behind each design decision. An HLD answers three core questions: WHAT are we building? HOW is it structured? WHY did we make these choices?

1.2 Intended Audience

This document is intended for academic evaluators, software developers, DevOps engineers, and technical reviewers involved in assessing or extending the project. It is written to be accessible even to readers who are new to Kubernetes or Laravel, using plain-English analogies alongside technical details.

1.3 System Overview

We are building a web-based Automotive Management System — a software tool that helps a car dealership or automotive service centre manage its day-to-day operations. Instead of paper records or spreadsheets, this system provides a clean web interface where staff can add, update and search vehicle inventory; manage customer records; book and track service appointments; generate sales and service reports; and control role-based access for admin, sales, and technician users.

Beyond building a web application, the project goes a step further by deploying it on Kubernetes, enabling automatic crash recovery, auto-scaling under load, and zero-downtime updates — mirroring real-world production engineering practices.

2. System Design

2.1 Application Design

The system follows a three-tier architecture. Tier 1 (Presentation) renders HTML pages via Laravel's Blade templating engine. Tier 2 (Application) contains all business logic — input validation, database queries, and access-control enforcement — running on Laravel with PHP-FPM. Tier 3 (Data) comprises MySQL for permanent structured storage and Redis for fast in-memory caching, session management, and background job queuing.

Laravel follows the Model-View-Controller (MVC) pattern: Models represent database entities (e.g., Vehicle, Customer), Views are Blade templates producing user-facing HTML, and Controllers orchestrate request handling between models and views.

2.2 Process Flow

A typical request flows as follows: (1) The user's browser sends an HTTP/HTTPS request to the server. (2) The Nginx Ingress Controller receives and routes the request. (3) Nginx forwards PHP requests to the Laravel PHP-FPM container. (4) Laravel checks user authentication via session data stored in Redis. (5) Laravel queries MySQL for data using the Eloquent ORM. (6) If the query result is cached in Redis, it is served immediately without a database round-trip. (7) Laravel renders a Blade template with the data. (8) The HTML response travels back through Nginx to the browser. (9) The user sees the rendered page.

2.3 Information Flow

All external traffic enters exclusively through the Nginx Ingress Controller (HTTPS). Internal services — MySQL and Redis — are exposed only as ClusterIP Services with no external IP address. Kubernetes NetworkPolicies enforce that only Laravel pods may communicate with MySQL and Redis. Configuration is provided through Kubernetes ConfigMaps (non-sensitive settings such as APP_URL and DB_HOST) and Secrets (sensitive values such as passwords and the Laravel APP_KEY), which are injected as environment variables at container startup.

2.4 Components Design

Laravel Application: The core PHP-FPM application container built on php:8.2-fpm-alpine, containing all business logic, routing, and Blade templates.

Nginx Web Server: Serves all static assets (CSS, JS, images) directly and proxies dynamic PHP requests to the Laravel container, separating concerns for improved performance.

MySQL Database (StatefulSet): Stores all persistent structured data in the automotive_db database. A PersistentVolumeClaim (mysql-pvc, 10Gi) ensures data survives container restarts.

Redis: An in-memory data store serving three roles — session storage (shared across multiple Laravel pods), query caching (reduces MySQL load), and queue backend (defers slow tasks like email sending to background workers).

Queue Worker: A separate Deployment running the same Laravel Docker image with the startup command php artisan queue:work, processing background jobs from the Redis queue without blocking web requests.

Scheduler CronJob: A Kubernetes CronJob that runs php artisan schedule:run on a scheduled interval to execute periodic tasks such as automated report generation.

2.5 Key Design Considerations

All resources are isolated within a dedicated Kubernetes Namespace (automotive-system). The Laravel Deployment runs with a minimum of 2 replicas for high availability, and a Horizontal Pod Autoscaler (HPA) automatically scales pods between 2 and 10 based on CPU utilisation, triggering at 70% average CPU. Kubernetes Rolling Update strategy ensures zero-downtime deployments. Alpine-based Docker images are used to minimise image size, reduce startup times, and limit attack surface.

2.6 API Catalogue

The system is a server-side rendered web application. Internal data access is handled through Laravel’s Eloquent ORM rather than a public REST API. Key application routes include: vehicle inventory CRUD operations, customer profile management, service booking scheduling and status updates, invoice generation and tracking, user authentication (login/logout), and role-based administrative functions. All routes are protected by Laravel middleware enforcing authentication and role-based access control.

3. Data Design

3.1 Data Model

All application data is stored in a MySQL relational database named automotive_db. The schema is managed entirely through Laravel Migrations, meaning the database structure is version-controlled alongside the application code.

Table Name	Key Columns	What It Stores
users	id, name, email, role, password	System users (admin, sales, technician)
vehicles	id, make, model, year, vin, price, status	Car inventory for the dealership
customers	id, name, email, phone, address	Customer profiles and contact details
service_bookings	id, customer_id, vehicle_id, date, status	Service appointment bookings
invoices	id, customer_id, amount, status, issued_at	Bills and payment tracking
audit_logs	id, user_id, action, entity, created_at	Record of every significant user action

3.2 Data Access Mechanism

All database access goes through Laravel's Eloquent ORM, which uses parameterised queries to prevent SQL injection. Frequently accessed datasets (such as the full vehicle list) are cached in Redis to reduce database load. Write operations (inserts/updates/deletes) bypass the cache and write directly to MySQL, with cache invalidation handled by the application logic.

3.3 Data Retention Policies

All MySQL data is persisted on a Kubernetes PersistentVolumeClaim (mysql-pvc, 10Gi), ensuring records survive container restarts, pod reschedules, and cluster updates. User-uploaded files, application logs, and compiled Blade views are stored on a separate PVC (laravel-storage-pvc, 5Gi) mounted at /var/www/html/storage. Redis data is in-memory only and is not persisted to disk; session and cache data is regenerated on restart.

3.4 Data Migration

Database schema changes are managed through Laravel Migrations. On first deployment, a Kubernetes InitContainer runs php artisan migrate automatically before the main Laravel container starts, ensuring the schema is always up to date. This approach version-controls the database structure alongside the application code, enabling repeatable and auditable schema evolution.

4. Interfaces

The primary user interface is a server-side rendered web application built with Laravel Blade templates and styled with Tailwind CSS, accessible via any modern browser over HTTPS. Users interact with role-specific dashboards, data tables, and forms. There is no separate mobile application; the interface is responsive and works on both desktop and mobile browsers.

External integrations are intentionally minimal in the current academic scope. The system integrates with Docker Hub for container image storage and with Let's

Encrypt (via cert-manager) for automated TLS certificate provisioning. The Nginx Ingress Controller serves as the single external-facing interface for all HTTP/HTTPS traffic.

5. State and Session Management

User sessions are stored in Redis rather than on the filesystem. This is critical for a multi-replica deployment: when the Horizontal Pod Autoscaler runs multiple Laravel pods simultaneously, each pod needs access to the same session data. By centralising sessions in Redis, any pod can serve any authenticated user's request transparently.

Session security is enforced through Laravel's built-in CSRF protection (hidden tokens in every form), encrypted session cookies, and configurable session lifetime. The Laravel APP_KEY (used for encryption) is stored in a Kubernetes Secret and injected as an environment variable, never appearing in source code or Docker images.

6. Caching

Redis serves as the primary caching backend. Frequently queried, read-heavy data (such as the full vehicle inventory list) is stored in Redis with a configurable TTL. When a user requests such data, Laravel first checks the Redis cache; on a cache hit, the result is returned in microseconds without a MySQL round-trip. On a cache miss, the data is fetched from MySQL and written to Redis for subsequent requests.

Nginx handles a second layer of caching by serving static assets (CSS, JavaScript, images) directly from the container filesystem, bypassing Laravel entirely. This eliminates PHP processing overhead for assets that never change between requests.

7. Non-Functional Requirements

7.1 Security Aspects

No Hardcoded Credentials: All sensitive values (database password, Laravel APP_KEY, Redis auth token) are stored in Kubernetes Secrets and injected as environment variables at runtime. They never appear in source code, Docker images, or version control.

Network Isolation: Only the Nginx Ingress is reachable from the internet. MySQL and Redis are ClusterIP services with no external IP. Kubernetes NetworkPolicies restrict MySQL and Redis access exclusively to Laravel pods.

Application Security: Laravel provides CSRF protection (hidden form tokens), bcrypt password hashing (passwords never stored in plain text), role-based access control via Gates and Policies (admin, sales, technician roles with distinct permissions), SQL injection prevention via Eloquent ORM parameterised queries, XSS prevention via automatic Blade output escaping, and full input validation through Laravel Form Requests.

TLS Encryption: All browser-to-server traffic is encrypted via HTTPS. The Nginx Ingress Controller handles TLS termination, with SSL certificates automated by cert-manager using the free Let’s Encrypt certificate authority. HTTP traffic is automatically redirected to HTTPS.

7.2 Performance Aspects

Quality Area	**Requirement**	**How We Achieve It**	**Explanation**
Availability	99.5% uptime	Multi-replica pods + K8s self-healing	If one replica crashes, others keep serving
Scalability	500+ concurrent users	HPA scales pods 2–10 on CPU	More users = more pods, automatically
Performance	Page load under 2 seconds	Redis caching + Nginx static serving	Cached results bypass slow DB calls
Security	No hardcoded credentials	All secrets in Kubernetes Secrets	Passwords never in code or images
Recoverability	Recovery within 5 minutes	Pod auto-restart + PVC data persistence	App auto-heals; data survives crashes
Portability	Runs on any K8s cluster	Standard YAML manifests, no cloud lock-in	Works on Minikube, GKE, EKS, AKS

8. References

1. Laravel Documentation v10.x: https://laravel.com/docs/10.x

2. Kubernetes Official Documentation v1.28: https://kubernetes.io/docs/

3. Docker Documentation: https://docs.docker.com/

4. Redis Documentation v7.0: https://redis.io/docs/

5. MySQL 8.0 Reference Manual: https://dev.mysql.com/doc/refman/8.0/en/

6. Nginx Documentation: https://nginx.org/en/docs/

7. cert-manager Documentation v1.13: https://cert-manager.io/docs/

8. Project Source Code: https://github.com/YashEC-19/laravel-k8s-automotive