Assigement (Moveinsync)

Name: Bhavesh Khokhar
College: Indian Institute of Information Technology Sonepat

Tech Stack Use:

Backend : Java Spring Boot

Frontend: React.js

Database: Mysql

Case Study : Intelligent Alert Escalation & Resolution System

Tables: Alert , Alert_transition, Alert_rule, Driver, Vechicle, Admin

Admin table:

```java
package com.example.demo.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data   9 usages   Bhaveshkhokhar
@Entity
@NoArgsConstructor
@AllArgsConstructor
public class Admin {
    @Id
    private String userName;
    private String password;

}
```

Driver table:

```java
@Id
@Column(name = "driver_id")
private String driverId;
private String driverName;
private String email;
private String phone;
private String status = "ACTIVE";
private LocalDateTime createdAt = LocalDateTime.now();
private LocalDateTime updatedAt = LocalDateTime.now();
```

Vechicle Table:

```
public class Vehicle {
    @Id
    @Column(name = "vehicle_id")
    private String vehicleId;
    private String registrationNumber;
    private String model;
    private String status = "ACTIVE";
    private LocalDateTime createdAt = LocalDateTime.now();
    private LocalDateTime updatedAt = LocalDateTime.now();
}
```

Alert_transition Table:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
@Column(name = "alert_id", nullable = false)
private Integer alertId;
@Column(name = "source_type", nullable = false)
private String sourceType;
private String type;
private String fromStatus;
private String toStatus;
@Column(columnDefinition = "TEXT")
private String metadata;
private LocalDateTime timestamp = LocalDateTime.now();
private LocalDateTime createdAt = LocalDateTime.now();
```

Alert table:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer alertId;
@Column(name = "source_type", nullable = false)
private String sourceType;
private String severity;
private String status;
private String type;
private Integer escalateCount;
@Column(name = "created_at")
private LocalDateTime createdAt = LocalDateTime.now();
@Column(name = "updated_at")
private LocalDateTime updatedAt = LocalDateTime.now();
@Column(name = "driver_id")
private String driverId;
@Column(name = "vehicle_id")
private String vehicleId;
@Column(columnDefinition = "TEXT")
private String resolutionComment;
@Column(columnDefinition = "TEXT")
private String metadata;
```

Admin_rule:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String ruleName;
private String alertType;
@Column(columnDefinition = "json")
private String conditions;
private String actionType;
private String targetSeverity;
private Boolean enabled = true;
private Integer priority = 100;
private LocalDateTime createdAt = LocalDateTime.now();
private LocalDateTime updatedAt = LocalDateTime.now();
```

Endpoints

- /api/alerts (post)
- /api/alerts (Get)
- /api/alerts/trends (Get)
- /api/alerts/weekly (Get)
- /api/alerts/{alertId} (Get)
- /api/alerts/{alertId}/resolve (Put)
- /api/alerts/{alertId}/history (Get)
- /api/alerts/auto-close (Get)
- /api/dashboard/stats (Get)
- /api/dashboard/top (Get)
- /login (Post)
- /api/rules (Get)
- /api/rules/reload (Get)

Authentication

I have use JWT and Spring security for Authentication

- JSON Web Token (JWT) is used to securely transmit user authentication data between the client and server.
- After successful login, the server generates a signed JWT and sends it to the client.
- The client includes this token in the header of subsequent requests for authorization.
- The server verifies the token's signature to ensure the request's authenticity and user identity
- code snippets

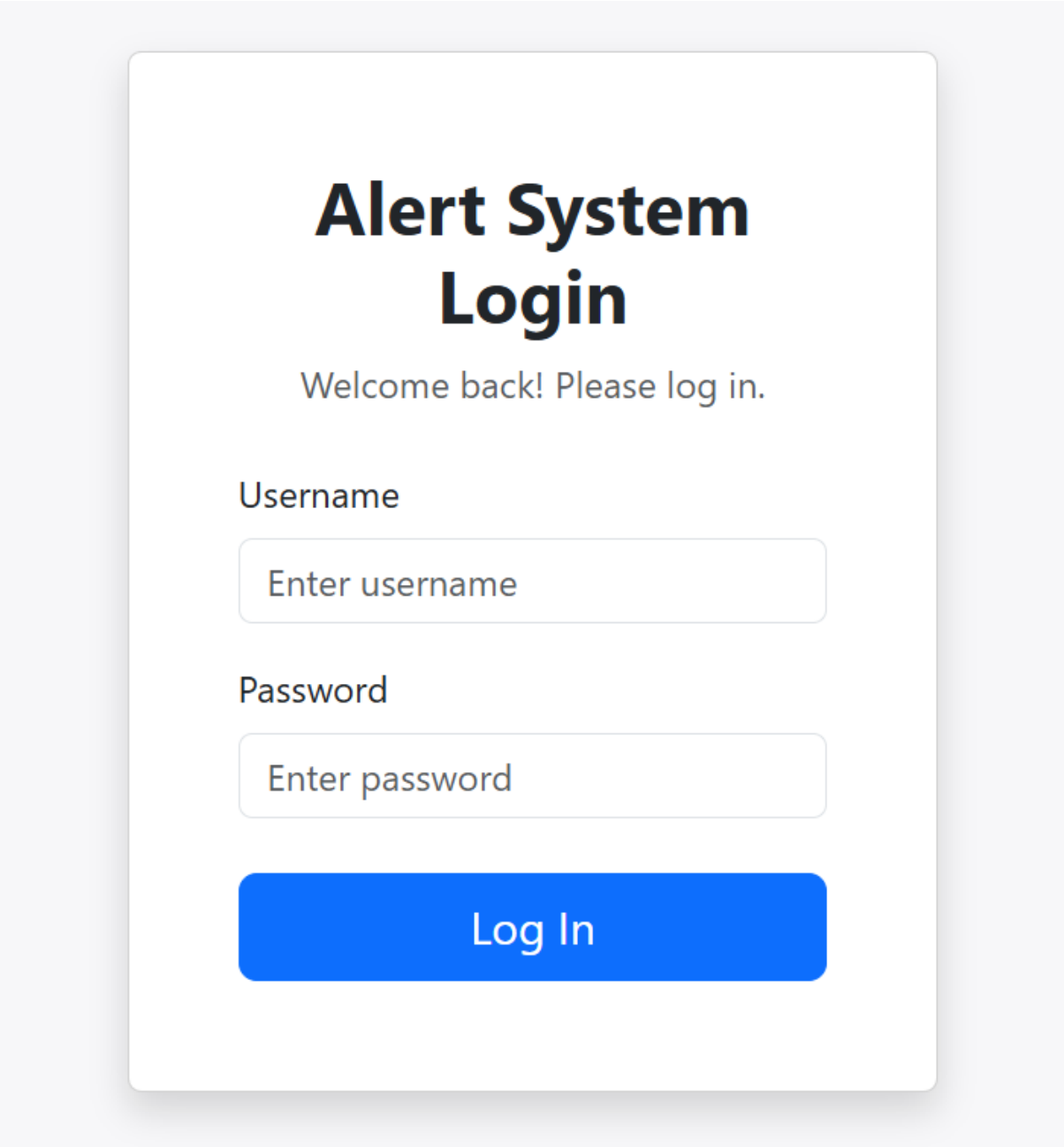
```
❖ public class SecurityConfig {
❖     @Autowired
❖     private UserDetailsService userDetailsService;
❖     @Autowired
❖     private JwtFilter jwtFilter;
❖     @Bean
❖     public AuthenticationProvider authenticationProvider(){
❖         DaoAuthenticationProvider provider=new DaoAuthenticationProvider();
❖         provider.setUserDetailsService(userDetailsService);
❖         provider.setPasswordEncoder(new BCryptPasswordEncoder(12));
❖         return provider;
❖     }
❖     @Bean
❖     public SecurityFilterChain securityFilterChain(HttpSecurity http)
  throws Exception {
❖         http.cors(Customizer.withDefaults())
❖                 .csrf(cutomizer->cutomizer.disable())
❖                 .authorizeHttpRequests((request->request
❖                         .requestMatchers("login","/actuator/prometheus")
❖                         .permitAll().anyRequest().authenticated()))
❖                 .httpBasic((Customizer.withDefaults()))
❖                 .sessionManagement(seesion-
  >seesion.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
❖                 .addFilterBefore(jwtFilter,UsernamePasswordAuthenticationFi
  lter.class);
❖         return http.build();
❖     }
❖     @Bean
```

```
❖     public CorsFilter corsFilter() {
❖         return new CorsFilter(corsConfigurationSource());
❖     }
❖
❖     private UrlBasedCorsConfigurationSource corsConfigurationSource() {
❖         CorsConfiguration config = new CorsConfiguration();
❖         config.setAllowedOrigins(List.of("http://localhost:5173"));
❖         config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE",
   "OPTIONS", "PATCH"));
❖         config.setAllowedHeaders(List.of("Authorization", "Content-Type"));
❖         config.setAllowCredentials(true);
❖
❖         UrlBasedCorsConfigurationSource source = new
   UrlBasedCorsConfigurationSource();
❖         source.registerCorsConfiguration("/**", config);
❖         return source;
❖     }
❖     @Bean
❖     public AuthenticationManager
   authenticationManager(AuthenticationConfiguration config) throws Exception
   {
❖         return config.getAuthenticationManager();
❖     }
❖ }
```

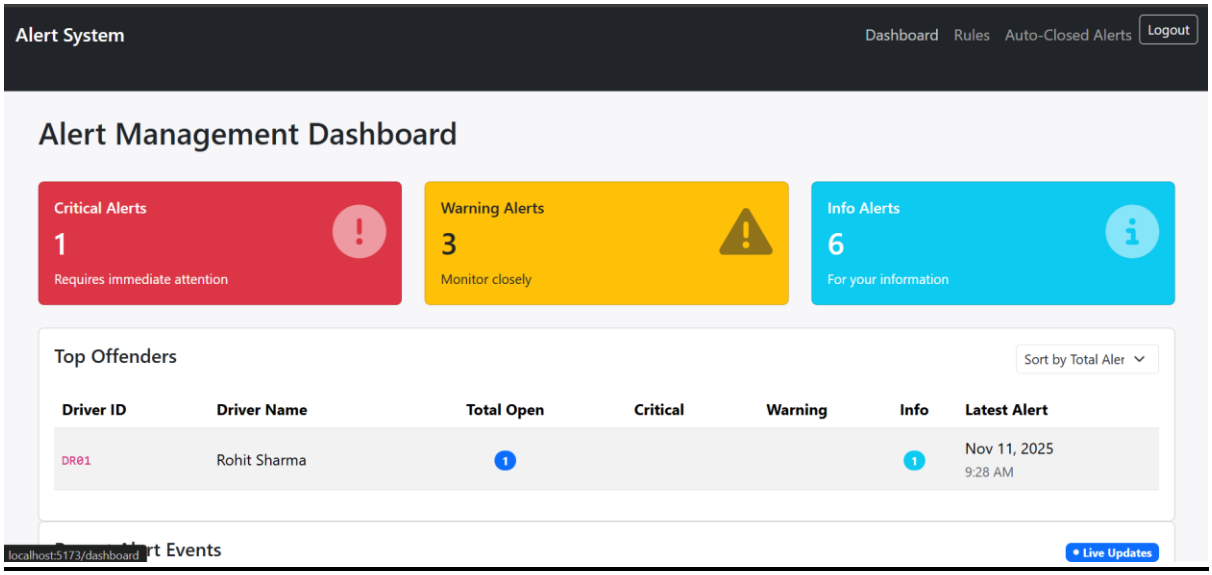
```
public class JwtFilter extends OncePerRequestFilter {
   @Autowired
   private JwtService jwtService;
   @Autowired
    ApplicationContext context;
    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
        String authHeader=request.getHeader("Authorization");
        String token=null;
        String userName=null;

        if(authHeader!=null&&authHeader.startsWith("Bearer ")){
            token =authHeader.substring(7);
            userName=jwtService.extractUserName(token);
        }
        if(userName!=null&&
SecurityContextHolder.getContext().getAuthentication()==null){
            UserDetails
userDetails=context.getBean(MyUserDetailsService.class).loadUserByUsername(use
rName);
```

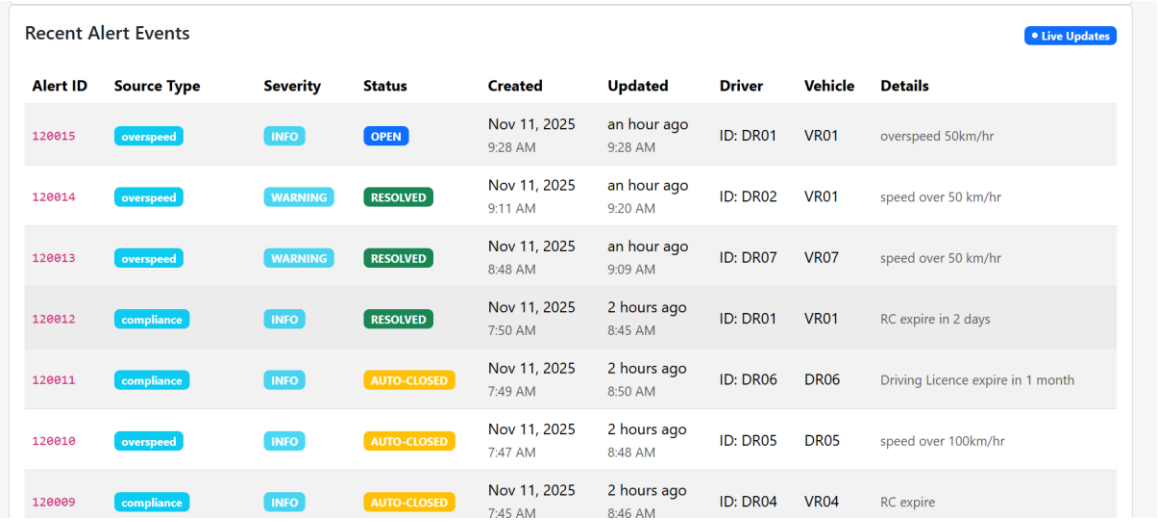
```
            if(jwtService.validateToken(token,userDetails)){
                UsernamePasswordAuthenticationToken authToken=new
UsernamePasswordAuthenticationToken(userDetails,null,userDetails.getAuthoritie
s());
                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken
);

            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Frontend Side Form:



DashBoard

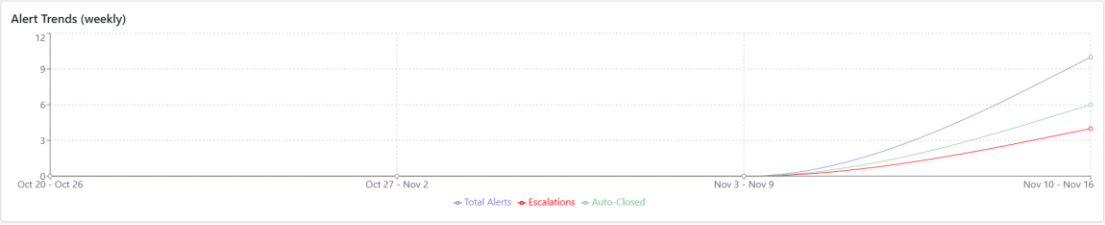




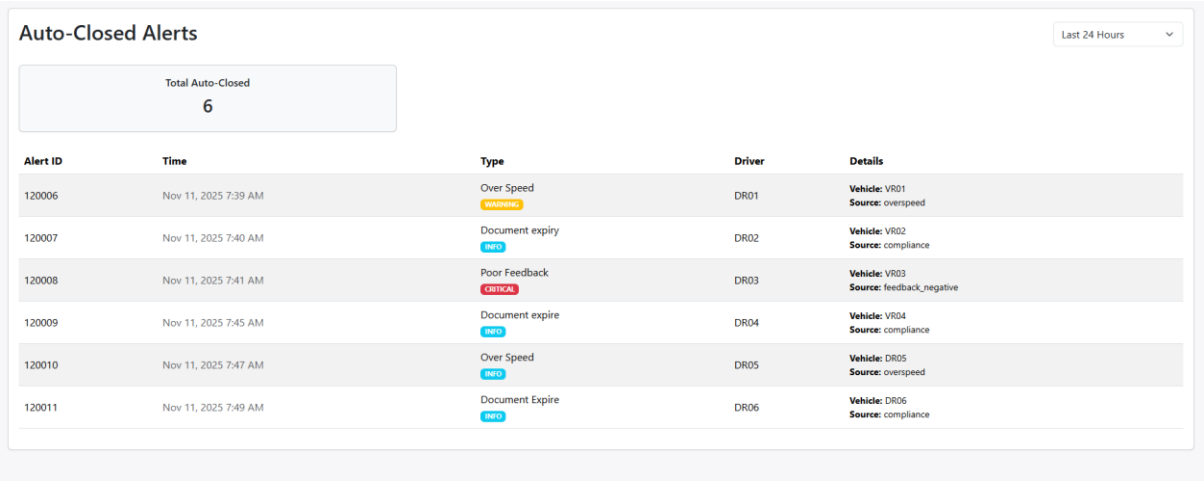
Recent Alert Events

Live Updates

Alert ID	Source Type	Severity	Status	Created	Updated	Driver	Vehicle	Details
120015	overspeed	INFO	OPEN	Nov 11, 2025 9:28 AM	an hour ago 9:28 AM	ID: DR01	VR01	overspeed 50km/hr
120014	overspeed	WARNING	RESOLVED	Nov 11, 2025 9:11 AM	an hour ago 9:20 AM	ID: DR02	VR01	speed over 50 km/hr
120013	overspeed	WARNING	RESOLVED	Nov 11, 2025 8:48 AM	an hour ago 9:09 AM	ID: DR07	VR07	speed over 50 km/hr
120012	compliance	INFO	RESOLVED	Nov 11, 2025 7:50 AM	2 hours ago 8:45 AM	ID: DR01	VR01	RC expire in 2 days
120011	compliance	INFO	AUTO-CLOSED	Nov 11, 2025 7:49 AM	2 hours ago 8:50 AM	ID: DR06	DR06	Driving Licence expire in 1 month
120010	overspeed	INFO	AUTO-CLOSED	Nov 11, 2025 7:47 AM	2 hours ago 8:48 AM	ID: DR05	DR05	speed over 100km/hr
120009	compliance	INFO	AUTO-CLOSED	Nov 11, 2025 7:45 AM	2 hours ago 8:46 AM	ID: DR04	VR04	RC expire



auto-close alert screen



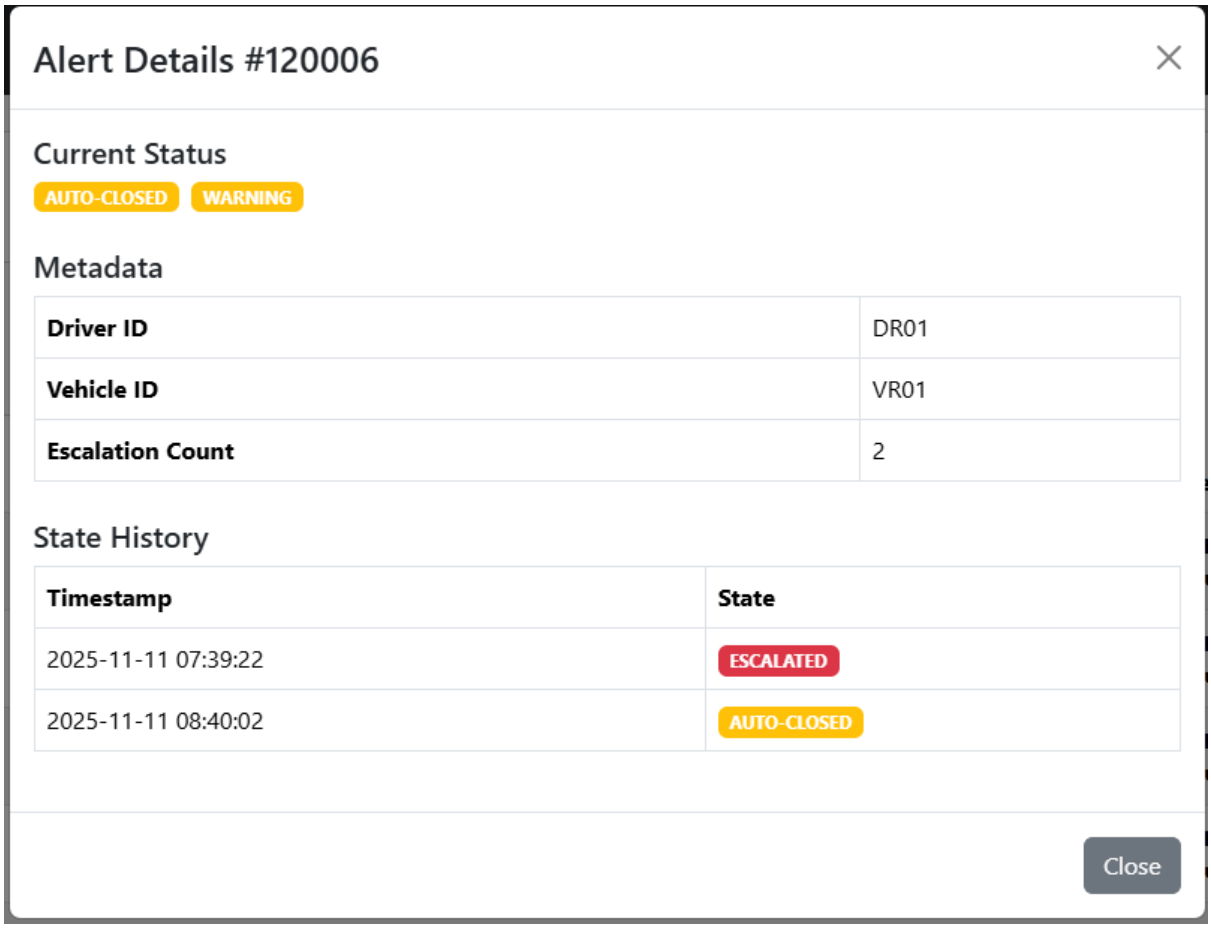
Auto-Closed Alerts

Last 24 Hours

Total Auto-Closed
6

Alert ID	Time	Type	Driver	Details
120006	Nov 11, 2025 7:39 AM	Over Speed WARNING	DR01	**Vehicle:** VR01 **Source:** overspeed
120007	Nov 11, 2025 7:40 AM	Document expiry INFO	DR02	**Vehicle:** VR02 **Source:** compliance
120008	Nov 11, 2025 7:41 AM	Poor Feedback CRITICAL	DR03	**Vehicle:** VR03 **Source:** feedback_negative
120009	Nov 11, 2025 7:45 AM	Document expire INFO	DR04	**Vehicle:** VR04 **Source:** compliance
120010	Nov 11, 2025 7:47 AM	Over Speed INFO	DR05	**Vehicle:** DR05 **Source:** overspeed
120011	Nov 11, 2025 7:49 AM	Document Expire INFO	DR06	**Vehicle:** DR06 **Source:** compliance

on click an alert show metadata



Alert Details #120006

Current Status

AUTO-CLOSED WARNING

Metadata

Driver ID	DR01
Vehicle ID	VR01
Escalation Count	2

State History

Timestamp	State
2025-11-11 07:39:22	ESCALATED
2025-11-11 08:40:02	AUTO-CLOSED

Close

If alert is not resolve or auto-closed then it show resolve button for manual resolution

Alert Details #120015

Current Status

OPEN INFO

Metadata

Driver ID	DR01
Vehicle ID	VR01
Escalation Count	0

State History

Timestamp	State
2025-11-11 09:28:19	OPEN

Resolve Alert Close

Admin rule

Rules Admin


```
{
  "safety": {
    "escalate_if_count": 3,
    "auto_close_window_hr": 1
  },
  "feedback": {
    "escalate_if_count": 2,
    "auto_close_window_hr": 24
  },
  "compliance": {
    "escalate_if_count": 1,
    "auto_close_if": "document_valid",
    "auto_close_window_hr": 1
  }
}
```


Reload Rules (server) Note: Edit server rules.json directly for permanent changes.

Trade-offs in the System

❖ **Monolithic Spring Boot Application vs Microservices Architecture**

Option	Pros	Cons	Decision
Monolithic (Current)	Easier to build and deploy; simpler development for a small team; all features in one codebase.	Harder to scale specific modules; a single failure can affect the whole app.	✅ Chosen for simplicity and maintainability during early stages.
Microservices	Independent scaling and deployment; better fault isolation.	Higher complexity (service discovery, communication, versioning).	Deferred for future — will migrate when system traffic increases.

❖ **Database Choice: MySQL vs NoSQL**

Option	Pros	Cons	Decision
MySQL (Relational)	Strong consistency, ACID compliance, good for structured alert data.	Scaling write-heavy workloads can be difficult.	✅ Chosen — ensures data integrity and supports complex queries (e.g., trends, aggregations).
NoSQL (MongoDB)	High scalability and flexible schema.	Weaker transactional support.	Deferred for future event-based analytics.

❖ **Real-Time Data Updates: WebSocket vs Polling**

Option	Pros	Cons	Decision
WebSocket (Push)	Real-time updates; efficient for live dashboards.	More complex backend setup; requires persistent connections.	✅ Will implement once traffic justifies.
Polling (Every few seconds)	Simple to implement using React setInterval.	Slightly higher latency; more network requests.	Currently used for simplicity.

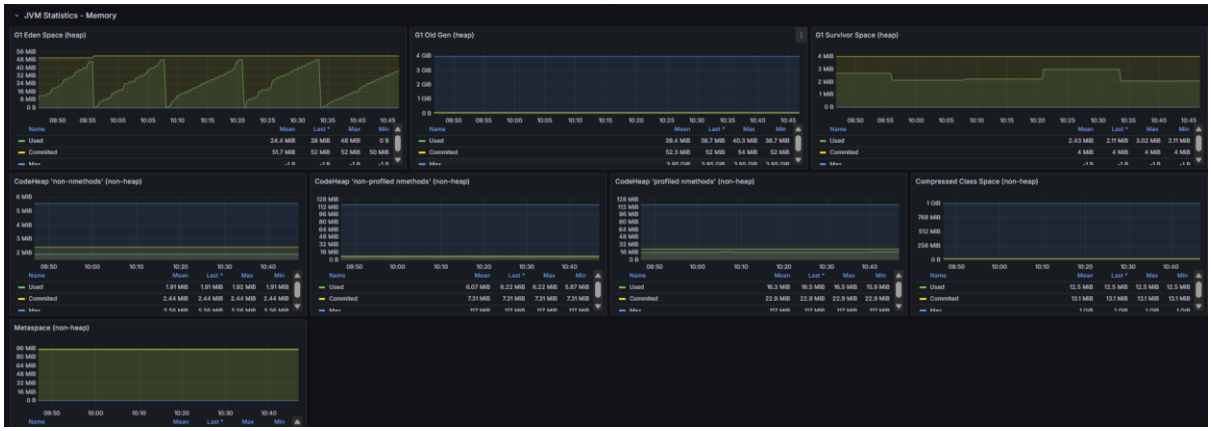
Error and Exception Handling:

for exception handling I have create global exception class also handle case special exception inside controller with try catch

```java
package com.example.demo.exception;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDateTime;
import java.util.Map;

@ControllerAdvice  no usages  Bhaveshkhokhar
public class GlobalExceptionHandler {
    @ExceptionHandler(RuntimeException.class)  no usages  Bhaveshkhokhar
    public ResponseEntity<?> handleRuntime(RuntimeException ex) {
        return ResponseEntity.badRequest().body(Map.of( k1: "timestamp", LocalDateTime.now(),  k2: "error", ex.getMessage()));
    }

    @ExceptionHandler(Exception.class)  no usages  Bhaveshkhokhar
    public ResponseEntity<?> handleAll(Exception ex) {
        return ResponseEntity.internalServerError().body(Map.of( k1: "timestamp", LocalDateTime.now(),  k2: "error", ex.getMessage()));
    }
}
```

System Monitoring :

- For system monitoring I have create Grafana dashboard
- I have use Prometheus which scratch data every 15 sec and Grafana show the metric also I have use log in my server to identify and address issues(Slf4j)
- Grafana Dashboard



Cost Estimation - Time and Space

Controller	Endpoint (HTTP Method & Path)	Purpose / Action	Time Complexity	Space Complexity	Explanation
AlertController	POST /api/alerts	Create new alert	**O(1)**	**O(1)**	Inserts one alert → single DB write.
	GET /api/alerts	Get latest 20 alerts	**O(1)**	**O(1)**	Fixed LIMIT 20 query → constant size result.
	GET /api/alerts/trends	Get daily alert trends (7 days)	**O(1)**	**O(1)**	Aggregation on 7-day window → constant.
	GET /api/alerts/trends/weekly	Get weekly alert trends (4 weeks)	**O(1)**	**O(1)**	Aggregation for 4 weeks → constant.
	GET /api/alerts/{alertId}	Get alert by ID	**O(1)**	**O(1)**	Indexed lookup by primary key.
	PUT /api/alerts/{alertId}/resolve	Resolve alert manually	**O(1)**	**O(1)**	Single record update.
	GET /api/alerts/{alertId}/history	Get alert lifecycle	**O(m)**	**O(m)**	Linear in number of transitions m for that alert.
	GET /api/alerts/auto-close	Get 50 auto-closed alerts	**O(1)**	**O(1)**	Fixed LIMIT 50 query → constant.
DashboardController	GET /api/dashboard/stats	Count alerts by severity	**O(1)**	**O(1)**	3 simple count queries using indexes.
	GET /api/dashboard/top	Get top 5 drivers	**O(1)**	**O(1)**	GROUP BY with LIMIT 5.
RulesController	GET /api/rules	Fetch all rule JSONs	**O(n)**	**O(n)**	Reads and parses all rule files; linear in number of rules.

Controller	Endpoint (HTTP Method & Path)	Purpose / Action	Time Complexity	Space Complexity	Explanation
	POST /api/rules/reload	Reload rule engine	**O(n)**	**O(1)**	Reloads all rules; linear in rule count.
UserController	POST /login	Authenticate admin and issue JWT	**O(1)**	**O(1)**	Single DB lookup + token generation.