FleetGuard360

Tutor

Diego Jose Luis Botia Valderrama

Responsables

Jeferson Alexander Del Rio Herrera

Davidson Arley Pérez Jiménez

Departamento de Ingeniería de Sistemas

Universidad de Antioquia

2025

1.Contexto del sistema

1.1. Descripción del sistema

El sistema consiste en un ecosistema de microservicios, basado en arquitectura Spring Boot, que permite gestionar de manera segura y eficiente las unidades de transporte de una organización, su localización en tiempo real y el historial detallado de sus movimientos.

Los microservicios se comunican a través de interfaces REST y están orquestados mediante un API Gateway, que centraliza las solicitudes y gestiona el acceso, y un Service Discovery que permite el registro dinámico y la detección de servicios. Cada microservicio se conecta a su propia base de datos PostgreSQL, siguiendo el principio de una base de datos por servicio para garantizar el desacoplamiento, la escalabilidad y la autonomía de cada componente.

1.2. Problemas que resuelve el sistema.

Problemática	Descripción
Falta de visibilidad en tiempo real	Permite conocer la ubicación exacta de cada unidad en cualquier momento, mejorando la planificación, el control y la respuesta ante incidentes.
Ausencia de trazabilidad histórica	Almacena el historial completo de movimientos de cada unidad, lo cual facilita auditorías y revisión de incidentes pasados.
Desorganización en la asignación de responsabilidades	Asocia cada unidad a un conductor y a una flota específica, permitiendo una gestión ordenada.
Riesgos de seguridad en el acceso	Implementa un microservicio de autenticación con generación de tokens JWT, asegurando que solo usuarios autorizados accedan a los datos y funcionalidades.

1.3. Usuarios del sistema.

Rol	Descripción
Administración	Configuran usuarios, supervisan el sistema completo, acceden a toda la información y definen reglas generales de operación.
Conductor	Responsable de operar unidades de transporte asignadas.
Unidad de transporte	Es la que envía información sobre su localización en tiempo real.

1.4. Diagrama de contexto.

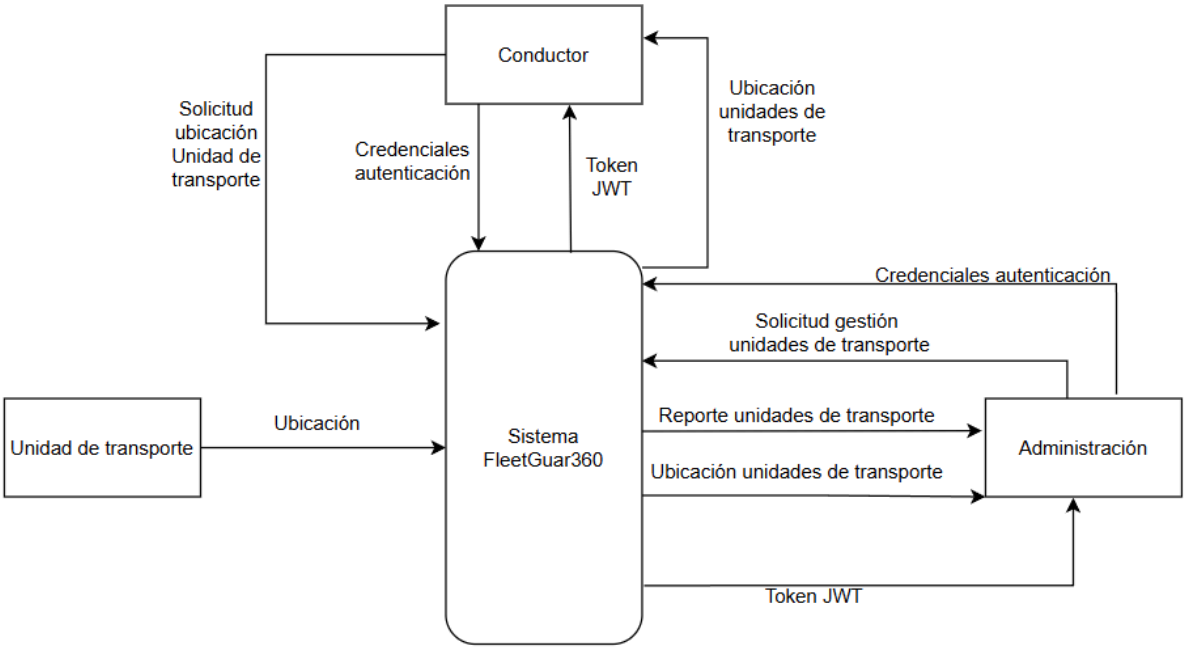


Figura 1. La imagen muestra el diagrama de contexto del sistema FleetGuar360, donde se visualizan las interacciones entre los actores externos (Conductor, Administración, Unidad de transporte) y el sistema.

2. Requisitos Arquitectónicos

2.1. Requisitos no funcionales.

A continuación se describen los principales requisitos no funcionales que debe cumplir el sistema FleetGuard360 para garantizar su correcto funcionamiento en entornos operativos reales, manteniendo niveles adecuados de seguridad, rendimiento, escalabilidad y mantenibilidad.

- **Seguridad**

El sistema debe implementar un mecanismo de autenticación basado en tokens JWT para todas las solicitudes protegidas.

Debe existir un sistema de autorización basado en roles, diferenciando al menos entre administradores, coordinadores y conductores.

Toda la comunicación entre los clientes, el API Gateway y los microservicios debe estar cifrada mediante HTTP.

El sistema debe incorporar medidas para prevenir vulnerabilidades comunes, incluyendo ataques de inyección SQL, cross-site scripting (XSS) y CSRF.

Las acciones relevantes del sistema deben ser registradas en un sistema de auditoría de accesos y operaciones.

- **Rendimiento**

El sistema debe ser capaz de responder a las consultas de localización actual con una latencia no superior a 500 milisegundos en condiciones normales de carga.

Las ubicaciones enviadas por las unidades de transporte deben ser procesadas y almacenadas en un tiempo máximo de 1 segundo.

Las interfaces deben ofrecer tiempos de respuesta consistentes para operaciones críticas como autenticación y visualización de unidades de transporte.

- **Escalabilidad**

La arquitectura basada en microservicios debe permitir el crecimiento independiente de cada componente, sin generar cuellos de botella globales.

El sistema debe ser capaz de soportar el crecimiento progresivo del número de unidades de transporte y usuarios sin degradar el rendimiento general.

- **Disponibilidad y resiliencia**

El sistema debe alcanzar un nivel mínimo de disponibilidad del 99% mensual.

Los microservicios deben ser tolerantes a fallos, de modo que la falla de un servicio no interrumpa la operación del resto del sistema.

- **Mantenibilidad**

El código del sistema debe seguir principios de diseño limpio y modular, facilitando la mantenibilidad y la evolución del software.

El sistema debe contar con un mecanismo de registro centralizado de logs que permita el monitoreo y la trazabilidad de eventos.

Se deben incluir herramientas de monitoreo de salud y métricas de rendimiento.

- **Portabilidad y despliegue**

El sistema debe ser desplegable en entornos basados en contenedores, como Docker, y orquestado mediante herramientas como Kubernetes.

El proyecto debe integrar un flujo de CI/CD que permita actualizaciones ágiles y controladas.

- **Persistencia y almacenamiento**

El sistema debe garantizar una gestión eficiente de grandes volúmenes de datos históricos, especialmente los relacionados con localización de unidades.

El sistema debe contar con mecanismos de respaldo periódico y recuperación ante desastres, permitiendo restaurar la operación en caso de pérdida de datos.

2.2. Restricciones técnicas.

Restricción	Descripción
Lenguaje de programación	Todo el backend debe desarrollarse en Java utilizando el framework Spring Boot.
Base de datos	El sistema debe utilizar PostgreSQL como sistema de gestión de base de datos.
Protocolo de comunicación.	Las comunicaciones entre clientes y servicios se realizarán mediante el protocolo HTTP, siguiendo un diseño RESTful.
Contenerización.	Todos los microservicios deben estar dockerizados para facilitar el despliegue, escalabilidad y portabilidad.
Plataforma de despliegue.	El sistema será desplegado en la plataforma Render.com.
Modelo de arquitectura.	Se adopta una arquitectura basada en microservicios, cada

	uno con su propia base de datos.
Gestión de base de datos en la nube.	Para persistencia en la nube, se debe emplear Supabase como servicio administrado de PostgreSQL.
Control de versiones.	El código fuente debe ser gestionado mediante Git con GitHub.
Autenticación.	La autenticación debe implementarse mediante JWT.
Monitoreo y observabilidad.	El sistema debe incluir un mecanismo de monitoreo con Prometheus para métricas, y Grafana para visualización y alertas.

3. Estilo y Patrones Arquitectónicos Seleccionados

3.1.Estilo Arquitectónico.

El sistema está diseñado siguiendo el estilo arquitectónico de microservicios, en el cual la funcionalidad del sistema se divide en múltiples servicios pequeños, autónomos y especializados. Cada microservicio implementa una única responsabilidad o dominio de negocio.

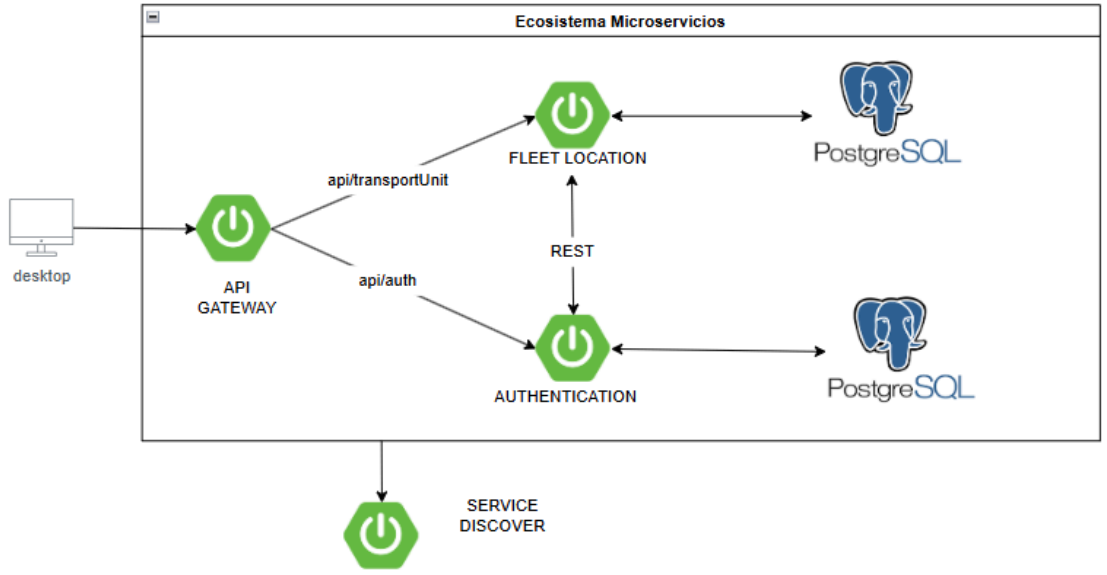


Figura 2. Esta imagen representa el ecosistema de microservicios del sistema FleetGuard360, compuesto por tres microservicios principales: API Gateway, Authentication y FleetLocation. Todos los servicios están registrados en un componente de Service Discovery, lo que permite su detección dinámica.

3.2. Patrón arquitectónico.

A continuación se muestran los principales patrones arquitectónicos utilizados para desarrollar el sistema de FleetGuard360.

Patrón	Descripción
API Gateway.	Punto de entrada único para todas las peticiones. Redirige, auténtica y aplica filtros de seguridad.
Service Discovery.	Permite el descubrimiento dinámico de servicios registrados.
Autenticación centralizada.	La seguridad se gestiona desde un microservicio de autenticación que emite tokens JWT.
Base de datos por servicio.	Cada microservicio gestiona su propio esquema y acceso a datos
Monitorización distribuida.	Se implementa mediante Prometheus y Grafana para recolectar métricas e identificar fallos o cuellos de botella.
Contenerización.	Cada servicio se ejecuta en su propio contenedor para garantizar aislamiento y portabilidad.

4. Vista de paquetes.

4.1. Diagrama de paquetes de alto nivel (modularización del sistema).

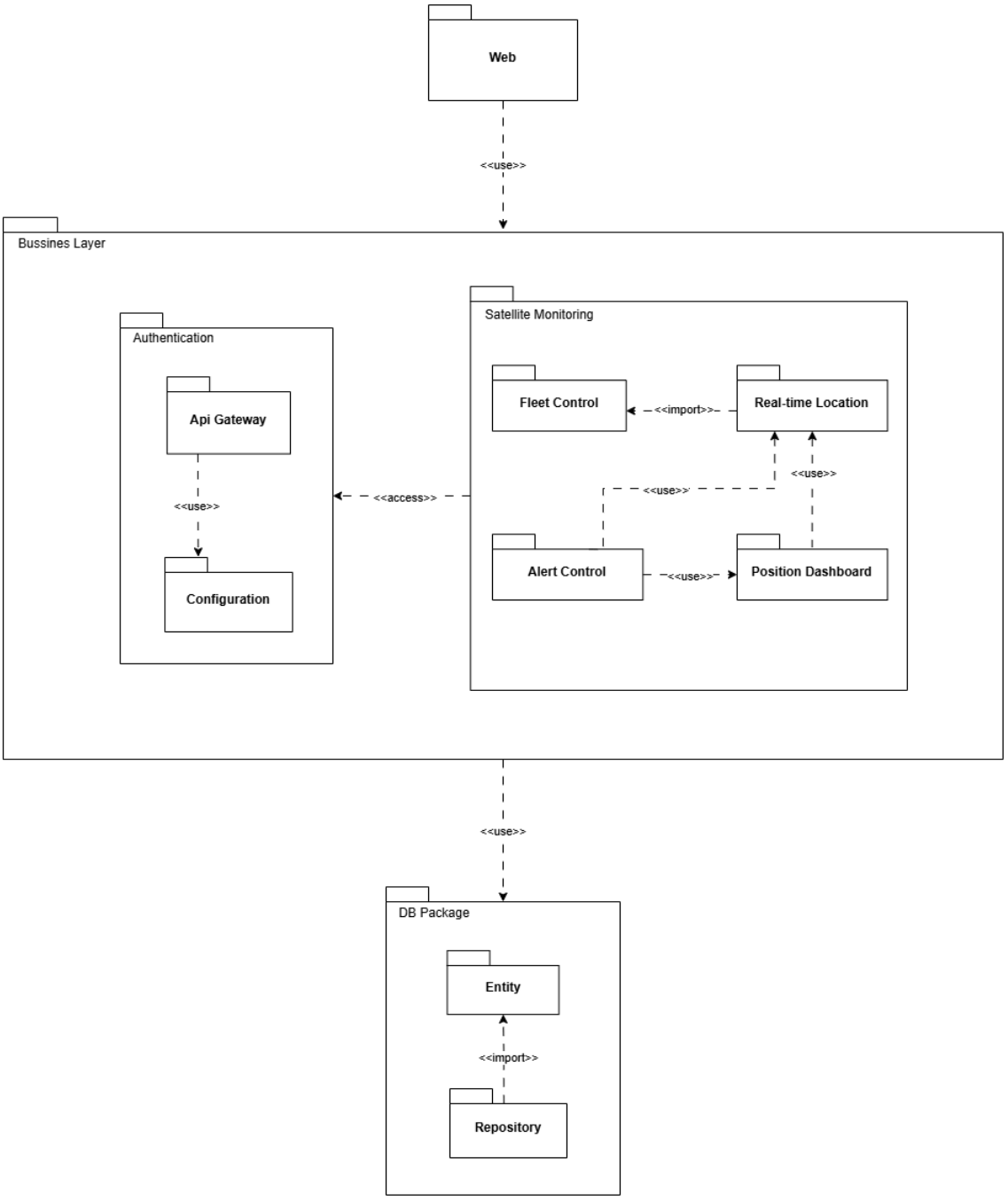


Figura 3. Esta imagen representa la modularización del sistema en paquetes.

4.2. Breve descripción de cada paquete.

- **Web:** También conocido como capa de presentación. Es el módulo con el cual interactúa el cliente, es decir, la interfaz de usuario y este debe usar la capa de negocio para la transferencia de información y su posterior persistencia.

- **Business Layer:** Es la capa de negocio, es decir, donde se implementará toda la lógica, separada en paquetes por cada microservicio. Esta capa usa el paquete DB para la persistencia del sistema.
    - **Authentication:** Es el paquete que contiene el acceso al sistema, es decir, el login y el registro de usuarios.
        - **Api Gateway:** Un paquete que contiene el servicio para la autorización. Ya que el sistema tiene usuarios con varios roles, es necesario validar los tokens generados por JWT para permitir el acceso a los endpoints dependiendo de los permisos del usuario. Este paquete utiliza el paquete Configuration para el correcto funcionamiento de la seguridad de la API.
        - **Configuration:** Es un paquete donde se encuentran las configuraciones de JWT y Spring Security.
    - **Satellite Monitoring:** Paquete en el que se encuentran todas las funcionalidades del servicio de monitoreo satelital de las flotas. Este paquete usa el de autenticación, ya que este último es necesario para el acceso y restricción del sistema dependiendo del rol y los permisos de cada usuario.
        - **Fleet Control:** Microservicio que se encarga del control de las flotas y su información, contiene funcionalidades de creación, edición y borrado de las unidades de transporte.
        - **Real-time Location:** Microservicio que se encargará de manejar los datos que llegan constantemente sobre la ubicación y velocidad de las unidades de transporte cada 15 segundos. Este paquete importa el de Fleet Control de transporte para acceder a la información de las unidades existentes.
        - **Alert Control:** Microservicio que se encarga de detectar excesos de velocidad de una unidad de transporte y generar las alertas o notificaciones correspondientes. Este paquete puede usar el de Real-time Location o el de Position Dashboard, ya que los dos contienen la información sobre la velocidad en cada instante de tiempo.
        - **Position Dashboard:** Microservicio que se encarga de guardar el historial de posiciones, velocidad y paradas de una unidad de transporte. Utiliza el paquete Real-time Location para obtener los datos mencionados en cada instante de tiempo.
- **DB Package:** Paquete que contiene el acceso a la base de datos por medio de Hibernate y JPA.

5. Vista de componentes.

5.1. Diagrama de componentes por módulo.

5.1.1 Diagrama de componentes módulo de autenticación.

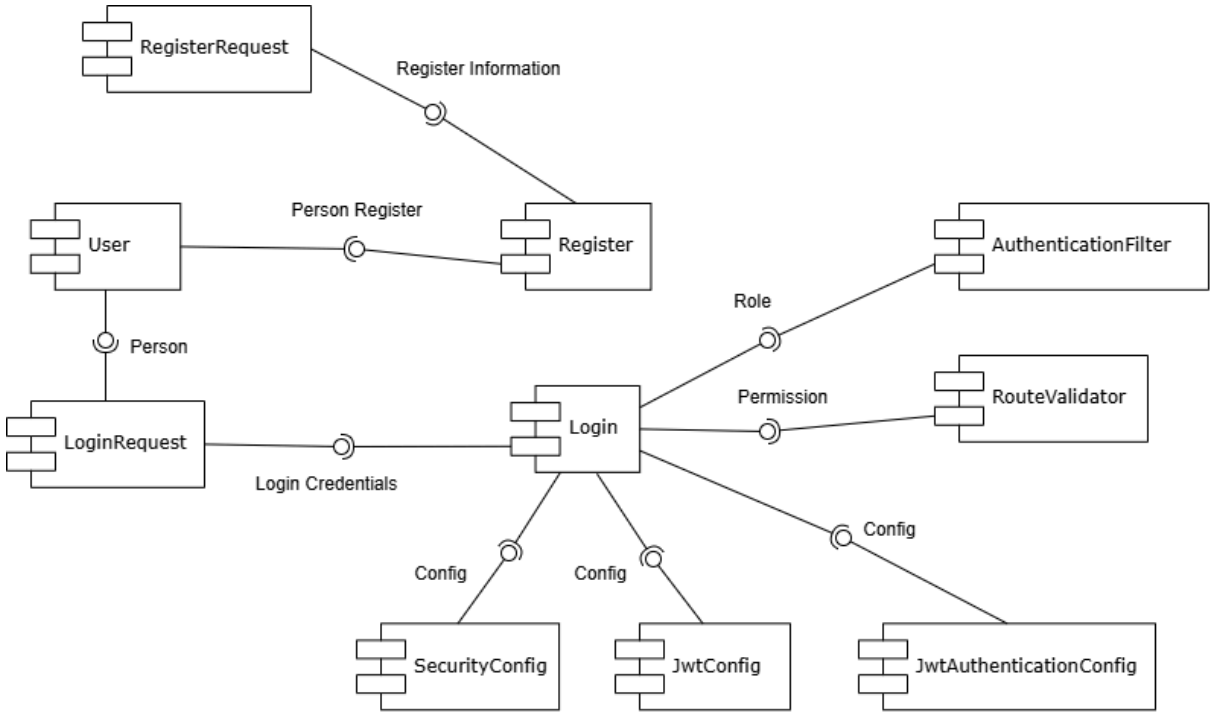


Figura 4. La imagen muestra el diagrama de componentes para el módulo de autenticación.

5.1.2 Diagrama de componentes módulo de monitoreo en tiempo real.

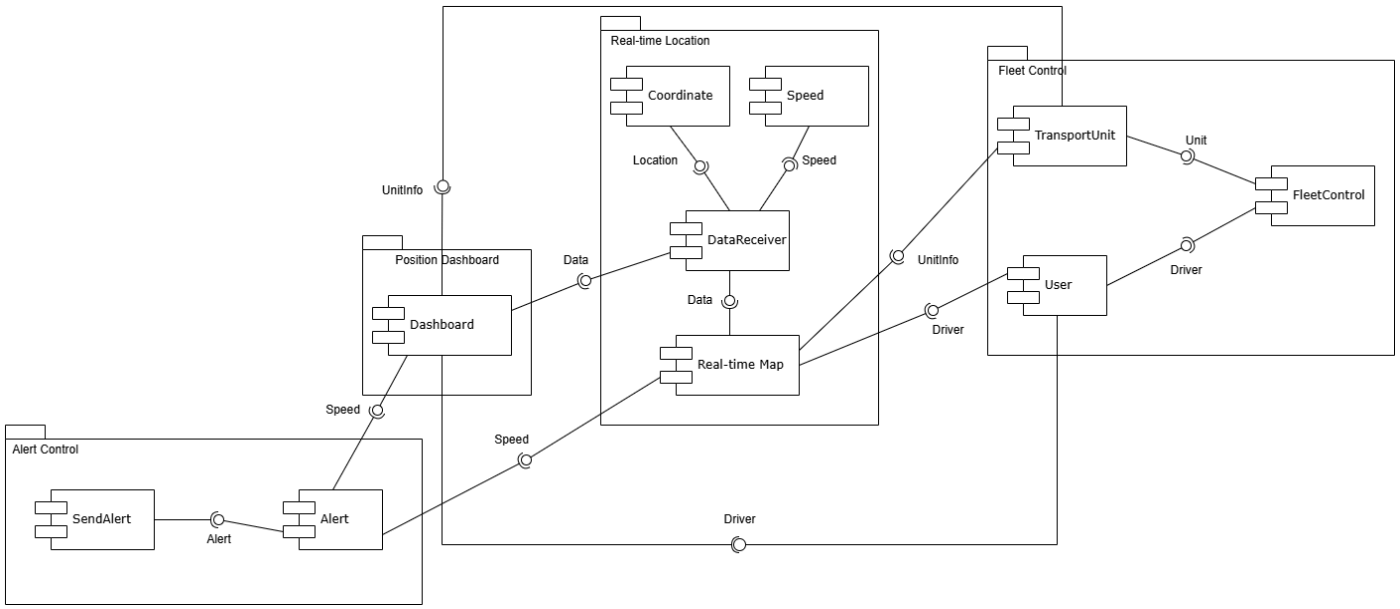


Figura 5. Imagen que muestra el diagrama de componentes para el módulo de monitoreo en tiempo real. Se involucran varios paquetes.

5.2. Descripción de responsabilidades de cada componente.

5.2.1. Módulo de autenticación.

Componente	Descripción
RegisterRequest	Contiene la información de registro de un nuevo usuario.
Register	Gestiona el registro de un nuevo usuario.
User	Contiene la información de un usuario registrado.
LoginRequest	Contiene las credenciales de inicio de sesión de un usuario registrado.
Login	Gestiona el inicio de sesión de un usuario, permitiendo o no su ingreso al sistema.
AuthenticationFilter	Valida el rol del usuario en el login para verificar sus permisos.
RouteValidator	Valida los permisos del usuario para saber a qué rutas o endpoints tiene permitido acceder.
SecutiryConfig	Configura Spring Security para la validación de rutas a las que un usuario puede acceder dependiendo de su rol y permisos.
JwtConfig	Configura el token JWT.
JwtAuthenticationConfig	Configura los permisos del usuario como autoridades en el token JWT.

5.2.2. Módulo de monitoreo en tiempo real.

Paquete	Componente	Descripción
Fleet Control	TransporUnit	Maneja la información sobre unidades de transporte.
	User	Maneja la información sobre conductores.
	FleetControl	Gestiona las operaciones sobre creación, edición y borrado de unidades de transporte.
Real-time Location	Coordinate	Recopila constantemente las coordenadas de una unidad de transporte.
	Speed	Recopila constantemente la velocidad de cada coordenada de una unidad de transporte.
	DataReceiver	Recopila los datos tanto de ubicación como de velocidad de una unidad de transporte.
	Real-time Map	Muestra o despliega en el mapa los datos recopilados de la ubicación en un instante de tiempo, así como su velocidad.

Position Dashboard	Dashboard	Muestra un historial de las posiciones y la velocidad en cada posición de una unidad de transporte, así como sus paradas.
Alert Control	Alert	Monitorea la velocidad y genera las alertas correspondientes.
	SendAlert	Envía las alertas generadas al conductor correspondiente.

5.3. Identificación de interfaces entre componentes.

5.3.1 Módulo de autenticación.

Proporciona	Requiere	Nombre
RegisterRequest	Register	Register Information
Register	User	Person Register
Person	LoginRequest	Person
LoginRequest	Login	Login Credentials
SecurityConfig	Login	Config
JwtConfig	Login	Config
JwtAuthenticationConfig	Login	Config
Login	RouteValidator	Permission
Login	AuthenticationFilter	Role

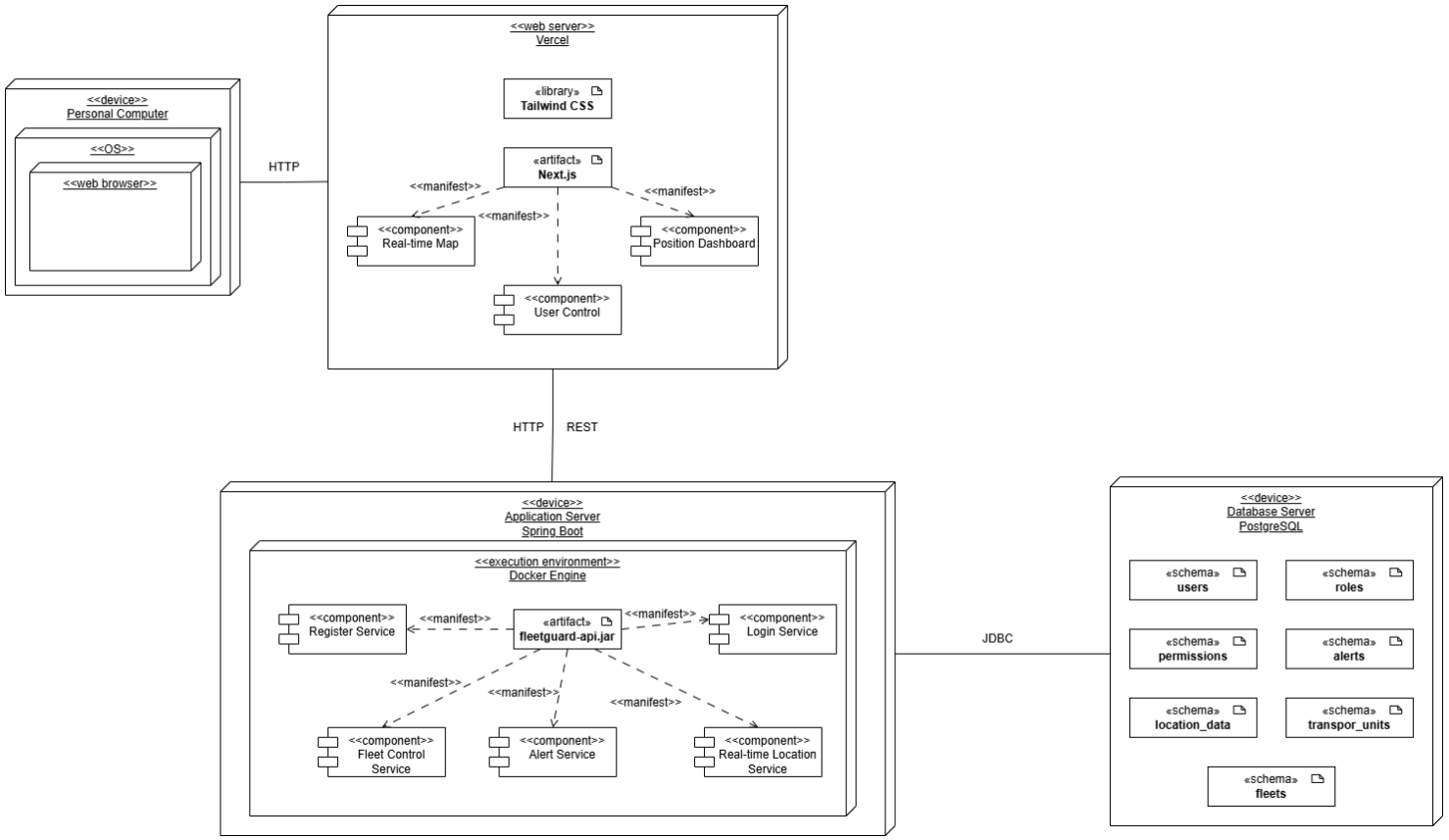
5.3.2. Módulo de monitoreo en tiempo real.

Proporciona	Requiere	Nombre
TransportUnit	FleetControl	Unit
TransportUnit	Real-time Map	UnitInfo
User	FleetControl	Driver
User	Real-time Map	Driver
Coordinate	DataReceiver	Location
Speed	DataReceiver	Speed
DataReceiver	Real-time Map	Data
DataReceiver	Dashboard	Data
Dashboard	Alert	Speed

Real-time Map	Alert	Speed
TransportUnit	Dashboard	UnitInfo
User	Dashboard	Driver
Alert	SendAlert	Alert

6. Vista de despliegue.

6.1. Diagrama de despliegue.



6.2. Tecnologías de despliegue previstas.

Tecnología	Descripción
Vercel	Simplifica el desarrollo, implementación y alojamiento de la aplicación web o frontend.
Tailwind CSS	Estilizado de componentes UI.
Next JS	Creado sobre React, es usado para la creación de la aplicación web.
Spring Boot	Es un framework de Java para el desarrollo web basado en microservicios (en nuestro caso).

Docker Engine	Plataforma de contenedores que aloja y aísla los microservicios.
PostgreSQL	Base de datos relacional.

7. Definición inicial de APIs.

Endpoint	Descripción
http://localhost:8080/apifleetLocation/ /user	Se obtienen todos los usuarios registrados.
http://localhost:8080/api/fleetLocation/ fleet	Se obtienen todas las flotas registradas.
http://localhost:8080/api/fleetLocation/ transportUnit	Se obtienen las unidades de transporte registradas que están activas.
http://localhost:8080/api/fleetLocation/transportUnit/update/{id}	Se actualiza la información de una unidad de transporte.
http://localhost:8080/api/fleetLocation/transportUnit/create	Se crea una nueva unidad de transporte.
http://localhost:8080/api/fleetLocation/ transportUnit/delete/{id}	Se hace una eliminación lógica de una unidad de transporte.
http://localhost:8080/api/auth/login	Se autentica un usuario en el sistema.
http://localhost:8080/api/fleetLocation/currentLocation	Se obtienen las ubicaciones actuales de las unidades de transporte que están activas.
http://localhost:8080/api/fleetLocation/currentLocation/create	Se registran las ubicaciones actuales que envían las unidades de transporte.
http://localhost:8080/api/fleetLocation/historiLocation/{id}	Se obtiene el historial de ubicaciones de una unidad de transporte.