UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

EcoScooter: Alquiler de Scooter con tecnología QR y GPS

Desarrollo Basado en Plataformas (CS2031) Laboratorio 1.02

INTEGRANTES

Nombre y Apellido	Codigo de Alumno	Participación
Juan Diego Azabache Liñan	202110430	100%
Ary Wener Aaron Rojas Durand	202310366	100%

Link del repositorio GitHub:
https://github.com/y0yoyopi/DemoScooterBackendDBP

DOCENTE: Mateo Noel Rabines

2024-01

01/06/2024

Resumen Ejecutivo

- Propósito del Proyecto:

El propósito de EcoScooter es ofrecer una solución de transporte urbano eficiente, conveniente y sostenible. Este tipo de sistema combina tecnología avanzada con un enfoque en la movilidad compartida, abordando varias necesidades y desafíos modernos

- Solución Propuesta:

La solución es un sistema completo que integra hardware avanzado, una aplicación móvil intuitiva, una infraestructura de soporte eficiente y un sistema de gestión centralizada. Este proyecto no solo mejora la movilidad urbana, sino que también promueve la sostenibilidad y la economía compartida, brindando a los usuarios una forma conveniente, segura y ecológica de desplazarse por la ciudad.

- Beneficios Clave:

Los beneficios clave del proyecto de alquiler de scooters con tecnología QR y GPS incluyen la mejora de la movilidad urbana, reducción del tráfico y de la huella de carbono, y mayor accesibilidad y conveniencia para los usuarios. Además, ofrece seguridad y monitoreo en tiempo real, facilitando el uso eficiente y sostenible de los scooters eléctricos en la ciudad.

Perfiles de Usuarios:

abarca individuos urbanos, generalmente entre 18 y 45 años, que buscan una solución de transporte rápida, económica y sostenible para desplazamientos cortos. Son personas tecnológicamente habilidosas, familiarizadas con el uso de aplicaciones móviles, y valoran la conveniencia de poder localizar, desbloquear y pagar Scooter fácilmente. Este perfil incluye estudiantes, profesionales, turistas y cualquier persona que prefiera evitar el tráfico y la búsqueda de estacionamiento, optando por una alternativa de movilidad ecológica y eficiente.

Funcionalidades de Búsqueda:

Las funcionalidades de búsqueda en el sistema de alquiler de scooters con tecnología QR y GPS permiten a los usuarios localizar rápidamente scooters disponibles, ver información detallada sobre el estado y disponibilidad de los scooters, planificar rutas eficientes, y recibir notificaciones sobre la disponibilidad en tiempo real, asegurando una experiencia de alquiler fluida y conveniente.

Reservas y Gestión de Solicitudes:

Las funcionalidades de reserva y gestión de solicitudes en el sistema de alquiler de scooters con tecnología QR y GPS ofrecen a los usuarios la capacidad de reservar scooters anticipadamente, gestionar y modificar sus reservas activas, y solicitar asistencia o reportar

incidencias de manera eficiente. Estas características mejoran la flexibilidad y la satisfacción del usuario, proporcionando una experiencia de uso más cómoda y confiable..

Seguridad y Confianza:

Las funcionalidades de seguridad y confianza en el sistema de alquiler de scooters con tecnología QR y GPS se centran en proteger la integridad de los usuarios, los scooters y la experiencia de uso en general. Desde la autenticación segura y el cifrado de datos hasta la prevención del robo de scooters y el soporte al cliente, estas características están diseñadas para promover la seguridad, la transparencia y la confianza en el servicio.

Facilidad de Uso:

la facilidad de uso en el proyecto se logra a través de un diseño intuitivo, procesos simplificados y acceso rápido a funciones clave, lo que garantiza que los usuarios puedan alquilar y utilizar scooters de manera rápida, cómoda y sin complicaciones

Introducción

Contexto del Proyecto: En respuesta a los desafíos de movilidad urbana y la necesidad de soluciones de transporte sostenible, se propone el desarrollo de una plataforma de alquiler de scooters con tecnología QR y GPS. Este proyecto busca abordar la congestión del tráfico y promover alternativas de movilidad más eficientes y respetuosas con el medio ambiente en entornos urbanos.

Propósito y Objetivos: El propósito principal de este proyecto es proporcionar a los usuarios una opción de transporte ágil, económica y sostenible para desplazamientos cortos en áreas urbanas. Los objetivos incluyen desarrollar una aplicación móvil intuitiva que permita a los usuarios ubicar, desbloquear y alquilar scooters fácilmente, garantizando al mismo tiempo la seguridad y la integridad de los usuarios y los vehículos.

Alcance del Proyecto: El alcance del proyecto abarca el diseño, desarrollo e implementación de una plataforma completa de alquiler de scooters, incluyendo una aplicación móvil para usuarios, sistemas de gestión de flotas, integración de tecnología QR y GPS en los scooters, y la implementación de medidas de seguridad y protección de datos.

Beneficios Esperados: Se espera que la implementación de esta plataforma proporcione beneficios significativos, como la reducción del tráfico vehicular, la mejora de la calidad del aire, la promoción de estilos de vida más activos y saludables, y la contribución a la mitigación del cambio climático al fomentar el uso de medios de transporte más sostenibles.

En resumen, este proyecto de alquiler de scooters con tecnología QR y GPS busca ofrecer una solución innovadora y eficiente para mejorar la movilidad urbana y promover un estilo de vida más sostenible en entornos urbanos densamente poblados.

Identificación del Problema o Necesidad
Descripción del Problema: En muchas ciudades, el crecimiento demográfico y el aumento del número de vehículos han generado problemas de congestión del tráfico y contaminación ambiental. Los sistemas de transporte público pueden resultar insuficientes o poco convenientes para desplazamientos cortos, mientras que el uso excesivo de vehículos privados contribuye al empeoramiento de la calidad del aire y la congestión de las calles.
Justificación:
Justificación del Proyecto: La implementación de una plataforma de alquiler de scooters con tecnología QR y GPS ofrece una solución viable y sostenible para abordar estos desafíos. Los scooters eléctricos son una alternativa de movilidad ágil, económica y respetuosa con el medio ambiente, que puede reducir la dependencia del automóvil y mejorar la accesibilidad al transporte en áreas urbanas.
Al integrar tecnología QR y GPS, la plataforma facilita a los usuarios localizar, desbloquear y alquilar scooters de manera rápida y eficiente a través de una aplicación móvil intuitiva. Además, al promover el uso compartido de scooters, se fomenta una cultura de movilidad más colaborativa y sostenible en la comunidad.

Descripción de la Solución:

La solución propuesta es desarrollar una plataforma de alquiler de scooters con tecnología QR y GPS. Esta plataforma consistirá en una aplicación móvil para usuarios, un sistema de gestión de flotas y una infraestructura de scooters equipados con tecnología QR y GPS.

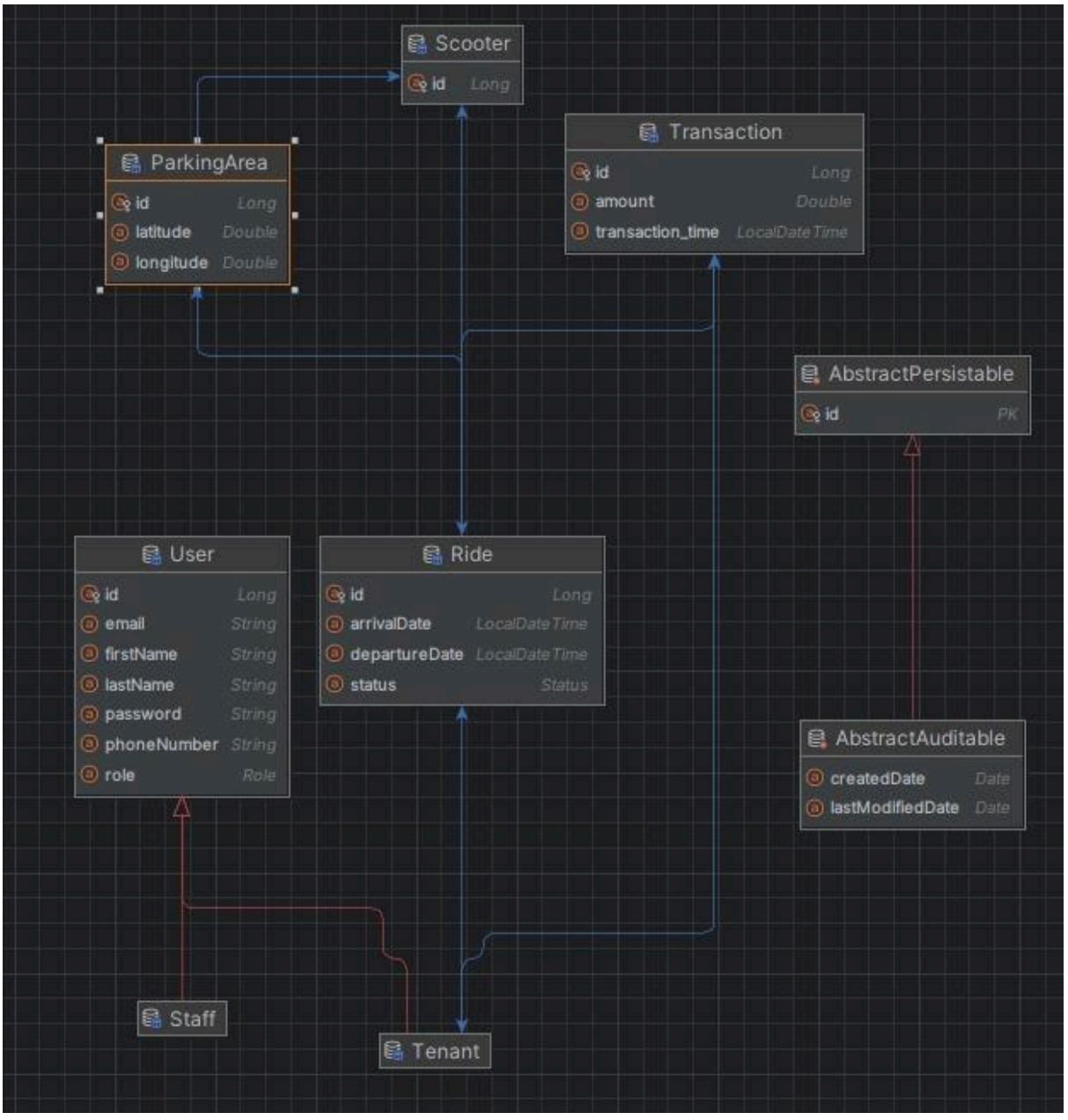
La aplicación móvil permitirá a los usuarios ubicar scooters disponibles en tiempo real, desbloquearlos mediante escaneo de códigos QR, y realizar el pago del alquiler de forma segura. Además, la aplicación proporcionará información sobre el estado de la batería y la ubicación del scooter, así como opciones para planificar rutas y recibir notificaciones relevantes.

El sistema de gestión de flotas permitirá a los operadores de la plataforma monitorear y administrar la flota de scooters de manera eficiente. Esto incluirá funciones para rastrear la ubicación de los scooters, gestionar la distribución y mantenimiento de la flota, y recopilar datos para análisis y mejora continua del servicio.

Los scooters estarán equipados con tecnología QR y GPS integrada para facilitar el alquiler y la gestión remota. Los usuarios podrán desbloquear los scooters escaneando un código QR con la aplicación móvil, mientras que el GPS permitirá rastrear la ubicación en tiempo real del scooter y garantizar su seguridad

Modelo Entidad-Relación

- Diagrama ER:



- Descripción de Entidades:

1. User

Atributos:
id: Representa el identificador único del usuario en la base de datos. Es de tipo Long y está anotado con @Id y @GeneratedValue para especificar que es una clave primaria generada automáticamente.

role: Representa el rol del usuario, que determina sus permisos y nivel de acceso en el sistema. Es de tipo Role, que parece ser una enumeración, y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos.

firstName: Representa el nombre del usuario. Es de tipo String y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos.

lastName: Representa el apellido del usuario. Es de tipo String y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos.

email: Representa el correo electrónico del usuario. Es de tipo String y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos.

password: Representa la contraseña del usuario. Es de tipo String y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos. Es obligatorio (nullable = false).

phoneNumber: Representa el número de teléfono del usuario. Es de tipo String y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos.

2. Transaction

Atributos:

id: Representa el identificador único de la transacción en la base de datos. Es de tipo Long y está anotado con @Id y @GeneratedValue para especificar que es una clave primaria generada automáticamente.

money: Representa la cantidad de dinero involucrada en la transacción. Es de tipo Double y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos. Es obligatorio (nullable = false).

transaction_time: Representa la fecha y hora en que se realizó la transacción. Es de tipo LocalDateTime y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos. Es obligatorio (nullable = false).

3. StaffAtributos

id: Identificador único del usuario.

role: Rol del usuario.
firstName: Nombre del usuario.
lastName: Apellido del usuario.
email: Correo electrónico del usuario.
password: Contraseña del usuario.
phoneNumber: Número de teléfono del usuario.

4. Tenant
- Atributos:

id: Identificador único del inquilino, heredado de la clase User.
role: Rol del inquilino, heredado de la clase User.
firstName: Nombre del inquilino, heredado de la clase User.
lastName: Apellido del inquilino, heredado de la clase User.
email: Correo electrónico del inquilino, heredado de la clase User.
password: Contraseña del inquilino, heredado de la clase User.
phoneNumber: Número de teléfono del inquilino, heredado de la clase User.
transaction: Una relación de uno a uno con la clase Transaction, representando la transacción asociada al inquilino. Se utiliza la anotación @OneToOne y @JoinColumn para mapear esta relación en la base de datos.
rides: Una relación de uno a muchos con la clase Ride, representando los viajes asociados al inquilino. Se utiliza la anotación @OneToMany para mapear esta relación en la base de datos, con la propiedad mapped
By indicando el atributo en la clase Ride que mantiene la relación inversa.

5. Scooter
- Atributos:

id: Identificador único del scooter. Es de tipo Long y está anotado con @Id y @GeneratedValue para especificar que es una clave primaria generada automáticamente.

rides: Una relación de uno a muchos con la clase Ride, representando los viajes asociados al scooter. Se utiliza la anotación @OneToMany para mapear esta relación en la base de datos, con la propiedad mappedBy indicando el atributo en la clase Ride que mantiene la relación inversa.

6. Ride
- Atributos:

id: Identificador único del viaje. Es de tipo Long y está anotado con @Id y @GeneratedValue para especificar que es una clave primaria generada automáticamente.

originName: Nombre del lugar de origen del viaje. Es de tipo String y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos. Es obligatorio (nullable = false).

destinationName: Nombre del lugar de destino del viaje. Es de tipo String y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos. Es obligatorio (nullable = false).

status: Estado del viaje, que parece ser una enumeración. Está anotado con @Column para mapearlo a la columna correspondiente en la base de datos. Es obligatorio (nullable = false).

departureDate: Fecha y hora de salida del viaje. Es de tipo LocalDateTime y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos.

arrivalDate: Fecha y hora de llegada del viaje. Es de tipo LocalDateTime y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos.

price: Una relación de uno a uno con la clase Transaction, representando el precio asociado al viaje. Se utiliza la anotación @OneToOne para mapear esta relación en la base de datos.

originParkingArea: Una relación de uno a uno con la clase ParkingArea, representando el área de estacionamiento de origen del viaje. Se utiliza la anotación @OneToOne para mapear esta relación en la base de datos.

destinationParkingArea: Una relación de uno a uno con la clase ParkingArea, representando el área de estacionamiento de destino del viaje. Se utiliza la anotación @OneToOne para mapear esta relación en la base de datos.

tenant: Una relación de muchos a uno con la clase Tenant, representando el inquilino asociado al viaje. Se utiliza la anotación @ManyToOne para mapear esta relación en la base de datos.

scooter: Una relación de muchos a uno con la clase Scooter, representando el scooter asociado al viaje. Se utiliza la anotación @ManyToOne para mapear esta relación en la base de datos.

7. Parking Area

- id: Identificador único del área de estacionamiento. Es de tipo Long y está anotado con @Id y @GeneratedValue para especificar que es una clave primaria generada automáticamente.

latitude: Latitud geográfica del área de estacionamiento. Es de tipo Double y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos. Es obligatorio (nullable = false).

longitude: Longitud geográfica del área de estacionamiento. Es de tipo Double y está anotado con @Column para mapearlo a la columna correspondiente en la base de datos. Es obligatorio (nullable = false).

scooters: Una relación de uno a muchos con la clase Scooter, representando los scooters estacionados en el área de estacionamiento. Se utiliza la anotación @OneToMany para mapear esta relación en la base de datos, con la propiedad mappedBy indicando el atributo en la clase Scooter que mantiene la relación inversa.

Testing

ParkingAreaControllerTest:

- Imports:

  Se importan diversas clases y bibliotecas necesarias para el funcionamiento del código, incluidas las relacionadas con Spring Boot, Mockito, JUnit, y Jackson.
  Anotaciones:

  @SpringBootTest: Indica que este es un test de Spring Boot que debería levantar el contexto completo de la aplicación.
  @AutoConfigureMockMvc: Configura automáticamente el MockMvc, que es una herramienta para realizar pruebas de controladores web.

- Inyección de Dependencias:

  @Autowired: Inyecta automáticamente las dependencias requeridas.
  @Mock: Indica que se está utilizando un mock (simulación) del ParkingAreaService.
  @InjectMocks: Inyecta los mocks en el ParkingAreaController.

- Objetos de Prueba:

  ParkingAreaResponseDto y CreateParkingAreaRequestDto: Objetos DTO utilizados para las respuestas y solicitudes de áreas de estacionamiento.
  MockMvc: Utilizado para simular llamadas HTTP a los endpoints del controlador.

- Método setUp:

  @BeforeEach: Método anotado para ejecutarse antes de cada prueba, inicializa los mocks y objetos de prueba.

- Pruebas:

  testCreateParkingArea: Verifica que la creación de un área de estacionamiento funcione correctamente.
  testGetParkingAreaById: Verifica que la obtención de un área de estacionamiento por ID funcione correctamente.
  testUpdateParkingArea: Verifica que la actualización de un área de estacionamiento funcione correctamente.
  testDeleteParkingArea: Verifica que la eliminación de un área de estacionamiento funcione correctamente.
  testGetAllParkingAreas: Verifica que la obtención de todas las áreas de estacionamiento funcione correctamente.
- Validaciones y Assertions:

  Se utilizan métodos de MockMvc como perform, andExpect, y result para realizar y validar las llamadas HTTP.

assertEquals se utiliza para comparar los valores esperados y reales en las respuestas.

RideControllerTest:

- Imports:

  Se importan diversas clases y bibliotecas necesarias para el funcionamiento del código, incluidas las relacionadas con Mockito, JUnit, Spring, y Jackson.

  Anotaciones:
  @Mock: Indica que se está utilizando un mock (simulación) del RideService.
  @InjectMocks: Inyecta los mocks en el RideController.

- Inyección de Dependencias:

  MockMvc: Utilizado para simular llamadas HTTP a los endpoints del controlador.
  RideService: Mock del servicio que maneja la lógica de negocio relacionada con los viajes.

- Objetos de Prueba:

  Tenant, Scooter, CreateRideRequestDto, RideResponseDto, RideDetailsDto, RidesByUserDto: Objetos de dominio y DTO utilizados para las solicitudes y respuestas de los viajes.

- Método setUp:

  @BeforeEach: Método anotado para ejecutarse antes de cada prueba, inicializa los mocks, el MockMvc y los objetos de prueba.

- Pruebas:

  testStartRide: Verifica que la creación de un viaje funcione correctamente.
  testCompleteRide: Verifica que la finalización de un viaje funcione correctamente.
  testCancelRide: Verifica que la cancelación de un viaje funcione correctamente.
  testGetRideById: Verifica que la obtención de un viaje por ID funcione correctamente.
  testGetRidesByTenant: Verifica que la obtención de todos los viajes de un inquilino funcione correctamente.
- Validaciones y Assertions:

  Se utilizan métodos de MockMvc como perform, andExpect, y result para realizar y validar las llamadas HTTP.
  Se utilizan validaciones de estado y tipo de contenido de la respuesta.

RideRepositoryTest:

- Imports:

  Se importan diversas clases y bibliotecas necesarias para el funcionamiento del código, incluidas las relacionadas con JUnit, Spring Data JPA, y las entidades del dominio.

- Anotaciones:

  @DataJpaTest: Indica que este es un test de Spring Data JPA que configura solo los componentes JPA y los arranca en una base de datos en memoria.
  @Autowired: Inyecta automáticamente las dependencias requeridas.

- Inyección de Dependencias:

  RideRepository rideRepository: Repositorio que maneja las operaciones de persistencia para los viajes.
  TestEntityManager entityManager: Proporciona una API para interactuar con la persistencia de entidades en pruebas.

- Objetos de Prueba:

  Tenant, Scooter, ParkingArea: Entidades del dominio que representan un inquilino (usuario), un scooter y un área de estacionamiento, respectivamente.

- Método setUp:

  @BeforeEach: Método anotado para ejecutarse antes de cada prueba, inicializa las entidades y persiste datos de prueba en la base de datos en memoria.

- Métodos Auxiliares:

  setupAndPersistTestRide(Status status): Crea y persiste un viaje con un estado específico.
  createTestRide(Status status): Crea una instancia de Ride con datos de prueba.

- Pruebas:

  testFindAllByTenantIdAndStatus: Verifica que se puedan encontrar todos los viajes de un inquilino con un estado específico.
  testCreateRide: Verifica que se pueda crear y persistir un viaje.
  testFindById: Verifica que se pueda encontrar un viaje por su ID.
  testDeleteById: Verifica que se pueda eliminar un viaje por su ID.

testFindAllByArrivalDateAndDestinationParkingArea: Verifica que se puedan encontrar todos los viajes por fecha de llegada y área de estacionamiento de destino.

- Validaciones y Assertions:

  Se utilizan métodos de JUnit como assertEquals, assertTrue, y assertFalse para validar los resultados de las operaciones de repositorio.
  PageRequest.of(0, 10): Utilizado para paginar los resultados de la búsqueda.

  ScooterControllerTest

  - Imports:

    Se importan diversas clases y bibliotecas necesarias para el funcionamiento del código, incluidas las relacionadas con JUnit, Mockito, Spring Boot Test, y las entidades del dominio.

  - Anotaciones:

    @SpringBootTest: Indica que este es un test de integración que arranca el contexto completo de Spring Boot.
    @AutoConfigureMockMvc: Habilita y configura MockMvc para pruebas de controladores web.
    @Mock: Crea e inyecta mocks de las dependencias necesarias.
    @InjectMocks: Crea una instancia del controlador y le inyecta los mocks creados con @Mock.
    @Autowired: Inyecta automáticamente las dependencias requeridas.

  - Objetos de Prueba:

    ScooterResponseDto y CreateScooterRequestDto: DTOs utilizados para simular solicitudes y respuestas en las pruebas.

  - Método setUp:

    @BeforeEach: Método anotado para ejecutarse antes de cada prueba, inicializa los DTOs de prueba y configura los mocks.

  - Pruebas:

    testCreateScooter: Prueba la creación de un scooter verificando la respuesta del endpoint /scooter.

testGetScooterById: Prueba la obtención de un scooter por su ID verificando la respuesta del endpoint /scooter/{scooterId}.
testUpdateScooter: Prueba la actualización de un scooter verificando la respuesta del endpoint PUT /scooter/{scooterId}.
testDeleteScooter: Prueba la eliminación de un scooter verificando la respuesta del endpoint DELETE /scooter/{scooterId}.
testGetAllScooters: Prueba la obtención de todos los scooters verificando la respuesta del endpoint /scooter.

- Validaciones y Assertions:

  Se utilizan métodos de JUnit como assertEquals para validar los resultados de las operaciones del controlador.
  Se utiliza MockMvc para simular y realizar peticiones HTTP a los endpoints del controlador.

- Referencias:
- Ppts del curso.
- Recursos de los E2E pasados.
- https://devdocs.io/javascript/
- Materiales de los repasos y asesorías del curso.
- Postman API Fundamentals Student Expert
- https://www.baeldung.com/jpa-cascade-types
- Herramientas como ChatGPT o Github Copilot.
- https://docs.spring.io/spring-data/jpa/reference/jpa/query-met hods.html
- https://learning.postman.com/docs/introduction/overview/
- https://documenter.getpostman.com/view/15567703/UVyxRtng
- Prevent Cross-Site Scripting (XSS) in a Spring Application | Baeldung