Parcial - Segundo Corte: Desarrollo de Aplicaciones Móviles

Construcción del Backend - Parte 1

- En este parcial, abordaremos la construcción del backend de una aplicación móvil. A continuación, se presenta un ejemplo funcional para orientar el desarrollo.
- Enlace Swagger:

```
http://localhost:9000/shopping-car/swagger-ui/index.html#/
```

Nota: - Si no se usa swagger, entonces adjuntar .json, con la documentación de las APIS. - Asignar el nombre `moviles_corte2` a la base de datos - Asignar el `port` - `9000`

Ejemplo:

- application.properties

```
server.servlet.context-path=/shopping-car
spring.application.name=shopping-car
server.port=
spring.jpa.hibernate.ddl-auto = update
spring.datasource.url = jdbc:mysql://localhost:3306/
spring.datasource.username =
spring.datasource.password =
```

- Entity

```
@Entity
@Table(name = "factura")
public class Factura {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "fecha", nullable = false)
    private LocalDateTime fecha;
    @ManyToOne(fetch = FetchType.EAGER, optional = false)
    @JoinColumn(name = "cliente_id", nullable = false)
    private Cliente cliente;
    //Encapsulamiento
}
```

- IRepository

```
@Repository
public interface IFacturaRepository extends JpaRepository<Factura, Long> {

}
```

- IService

```
public interface IFacturaService {
    List<Factura> findAll();
    Optional<Factura> findById(Long id);
    Factura save(Factura factura);
    void update(Factura factura, Long id);
    void delete(Long id);
}
```

- Service

```
@Service
public class FacturaService implements IFacturaService {

    @Autowired
    private IFacturaRepository repository;
    @Override
    public List<Factura> findAll() {
    return repository.findAll();
    }
    @Override
    public Optional<Factura> findById(Long id) {
        return repository.findById(id);
    }
    @Override
    public Factura save(Factura factura) {
        return repository.save(factura);
    }
    @Override
    public void update(Factura factura, Long id) {
        Optional<Factura> ps = repository.findById(id);
        if (!ps.isEmpty()){
            Factura facturaUpdate = ps.get();
            facturaUpdate.setFecha(factura.getFecha());
            facturaUpdate.setCliente(factura.getCliente());
            repository.save(facturaUpdate);
        }else{
            System.out.println("No existe el factura");
        }
```

```
    }
    @Override
    public void delete(Long id) {
        repository.deleteById(id);
    }
}
```

- Controller

```
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("api/factura")
public class FacturaController {
    //Inyectar el servicio
    @Autowired
    private IFacturaService service;

    @GetMapping()
    public List<Factura> findAll() {
        return service.findAll();
    }
    @GetMapping("/{id}")
    public Optional<Factura> findById(@RequestParam Long id) {
        return service.findById(id);
    }
    @PostMapping()
    public Factura save(@RequestBody Factura factura) {
        return service.save(factura);
    }
    @PutMapping("/{id}")
    public void update(@RequestBody Factura factura, @RequestParam Long id) {
        service.update(factura, id);
    }
    @DeleteMapping("/{id}")
    public void delete(@RequestParam Long id) {
        service.delete(id);
    }
}
```

Construcción del FrontEnd - Parte 2

1. Instalar Ionic y Cordova:
    - Asegúrate de tener Node.js instalado en tu sistema e enstala Ionic y Cordova globalmente usando npm (Node Package Manager): `npm install -g @ionic/cli cordova`
2. Crear un nuevo proyecto Ionic, para ello utiliza Ionic CLI para crear un nuevo proyecto: `ionic start shopping-cart blank`