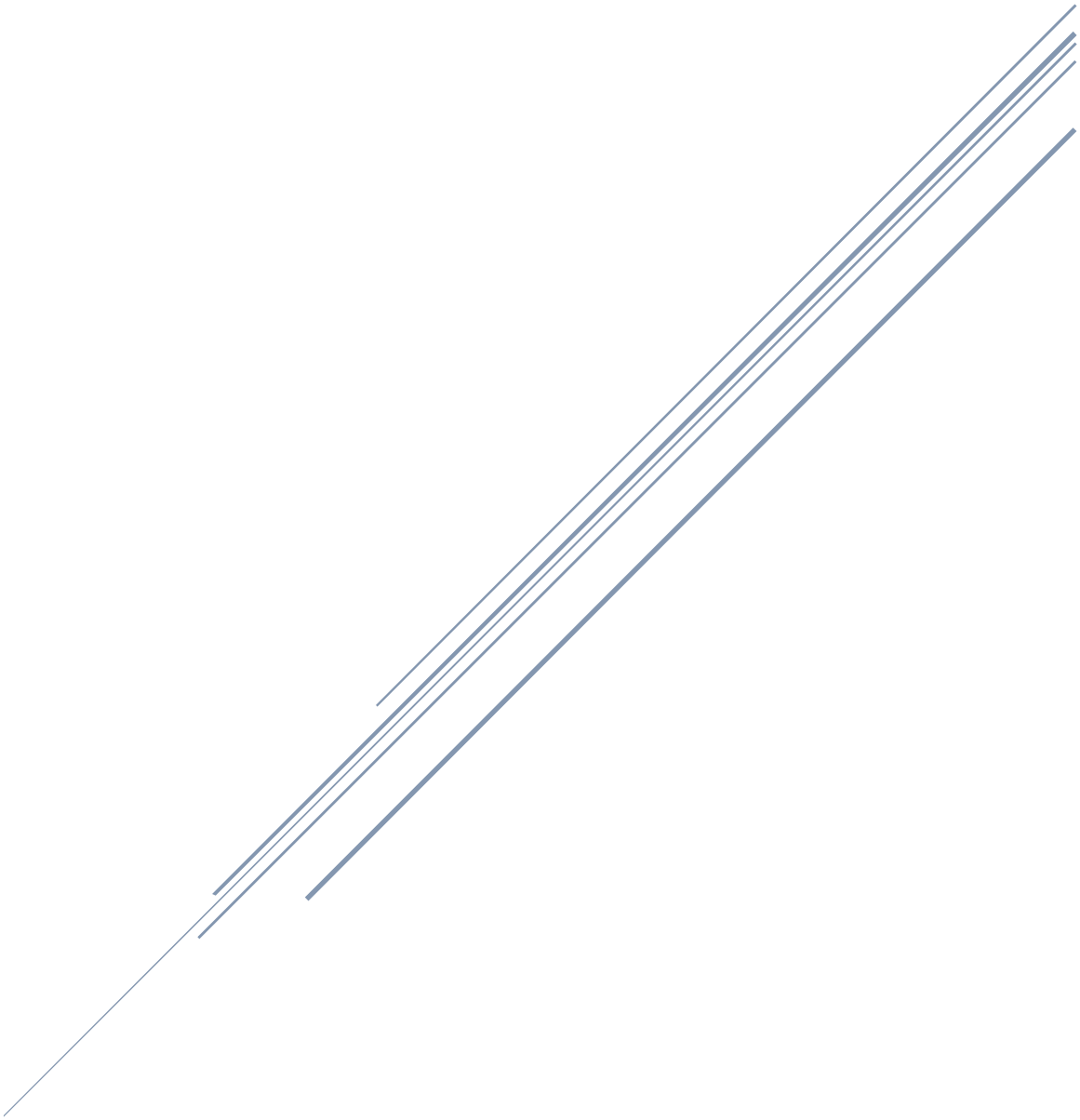
ARCHIVO TÉCNICO

Componente Punto de venta

PointOfSaleServer



Caot2204

ÍNDICE

1. Introducción

Este documento permite a diseñadores y desarrolladores conocer aspectos importantes del componente para el software Punto de Venta, tales como el comportamiento, interfaces, suposiciones y requerimientos necesarios para funcionar. Se ha realizado con la premisa de que cualquiera que necesite un punto de venta utilice este documento y sus respectivos archivos como punto de inicio para utilizarlo de manera directa o extenderlo a como lo necesite.

2. Propósito

El componente PointOfSaleServer(Punto de Venta) se ha realizado con la finalidad de que éste se encargue de la administración de la base de datos de una tienda desde un servidor, ya sea que éste se encuentre en la misma computadora o en un servidor remoto. El componente puede ser utilizado por clientes independientemente de la plataforma que sea, ya que se comunica mediante el protocolo JSON.

3. Archivo técnico

- **Nombre:** PointOfSaleServer.
- **Versión:** 1.0
- **Descripción:** Este software es utilizado para administrar la base de datos de una tienda desde un servidor local o remoto, permite la inserción, consulta, actualización, eliminación de información sobre inventario, ventas y usuarios con roles específicos. Está diseñado para su extensión.
- **Lenguaje (Language):** JavaScript
- **Variables de entrada (Input variables):**
  - loggingData: JsonObject = {
    userName: String(30) = “”
    password: String = “"
    }
  - category: JsonObject = {
    name: String(30) = “”
    }
  - product : JsonObject = {
    code: String(20) = “”
    name: String(100) = “”
    price: Double = 0.0
    stock: Integer = 0
    isInfinityStock: Boolean = false
    }

- user: JsonObject = {
  name: String = “”
  password: String = “”
  isAdmin: Boolean = false
  }

Decisión de diseño: Se a optado por almacenar el nombre y precio actual del producto con la finalidad de mantener la integridad de las ventas registradas, ya que si únicamente se relacionan las entidades, en consultas futuras de las ventas, pueden existir diferencias en los precios individuales y por tanto el monto total de la venta registrada.

Decisión de diseño: Se a optado por recibir la fecha de la venta por parte del software client, esto, para que coincida con la fecha en la que se imprima el ticket o en la que se guardo en el momento exacto de hacer la venta en la tienda, ya que por algún motivo, la fecha y hora puede variar en lo que la solicitud es recibida por el servidor.

- sale: JsonObject = {
  idPosUser: String(30) = “”,
  dateOfSale: Date = YYYY-MM-DD HH:MM:SS,
  productsSold: [
  {
  code: String(20) = “”,
  name: String(100) = “”,
  price: Double = 0.0,
  units: Integer = 1
  },
  ...
  ]
  }

- **Variables de salida (Output variables):**
  - inventory: JsonArray = [Product, Product, ...]
  - category: JsonObject = {
    id: Integer = 0,
    name: String(30) = “”
    }
  - product : JsonObject = {
    code: String(20) = “”
    name: String(100) = “”
    price: Double = 0.0

stock: Integer = 0
isInfinityStock: Boolean = false
}

- user: JsonObject = {
  id: Integer = 0,
  name: String = “”,
  password: String = “”,
  isAdmin: Boolean = false
  }
- tokenUserSession: { String = “” }
- sale: JsonObject = {
  saleData: {
  id: Integer = 0,
  dateOfSale: Date,
  User: {
  name: String = “”
  }
  }
  totalSale: Double = 0.0,
  productsSold: [
  {
  code: String(20) = “”,
  name: String(100) = “”,
  price: Double = 0.0,
  units: Integer = 1
  },
  ...
  ]
  }
- sales: JsonArray = {
  day: String = “YYYY-MM-DD”,
  totalSaleOfDay: Double = 0.0,
  salesOfDay: [Sale, Sale, ...]
  }

- **Suposiciones:**
  - El servidor tiene instalado Node.js para la ejecución del componente.

- El servidor tiene instalado algún manejador de base de datos; ya que el componente utiliza Sequelize, se puede usar uno de los manejadores de base de datos soportados por éste.

3.1 Variables de entrada

Name:	loggingData
Data type:	JsonObject
Description:	Datos a utilizar para el inicio de sesión del usuario.
Properties:	user_name: String(30) = “”
	password: String = “”

Name:	category
Data type:	JsonObject
Description:	Datos de una categoría para registrar productos.
Properties:	name: String(30) = “”

Name:	product
Data type:	JsonObject
Description:	Datos del producto a guardar en el inventario.
Properties:	code: Int = 0
	name: String = “”
	price: Double = 0.0
	stock: Int = 0
	isInfinityStock: Boolean = false/true

Name:	user
Data type:	JsonObject
Description:	Datos del usuario que utilizará el sistema, tendrá los roles que el desarrollador desee. En este componente tendra el rol de Administrador o Cajero.

	name: String(30) = “”
Properties:	password: String = “”
	isAdmin: Boolean = false

Name:	sale
Data type:	JsonObject
Description:	Datos de la venta la cual se registrará por el usuario (cliente). La propiedad productsSold puede contener 1 o más productos, deben seguir el mismo formato.
Properties:	dateOfSale: DateTime (Provista por el ORM), totalSale: productsSold: [{ code: String(20) = “”, name: String(100) = “”, price: Double = 0.0, units: Integer = 1 }, ...]

3.1 Variables de salida

Las variables de salida category, product, user y sale tienen los mismos datos que las variables de entrada con sus respectivos nombres, agregando el id generado por la API a cada uno de estos objetos, a excepción del objeto product.

Name:	inventory
Data type:	JsonArray
Description:	Inventario de la tienda. (Productos registrados en la base de datos)

Properties:

```
[
  {
   "code": "",
   "name": "",
   "price": ,
   "stock": ,
   "isInfinityStock": ,
   "Category": {
    "name": ""
   }
  },
  {
   "code": "",
   "name": "",
   "price": ,
   "stock": ,
   "isInfinityStock": ,
   "Category": {
    "name": ""
   }
  }, . . .
]
```

Name: tokenUserSession

Data type: String

Description: Token utilizado para autorizar operaciones en el componente.

Name: sales

Data type: JsonArray

Description: Ventas de la tienda registradas en la base de datos, filtradas por un día en específico, puede ser usada para verificar el corte de caja.

Properties:

```
{
  day: String = “YYYY-MM-DD”,
  totalSaleOfDay: Double = 0.0,
  salesOfDay: [ Sale, Sale, ... ]
 }
```

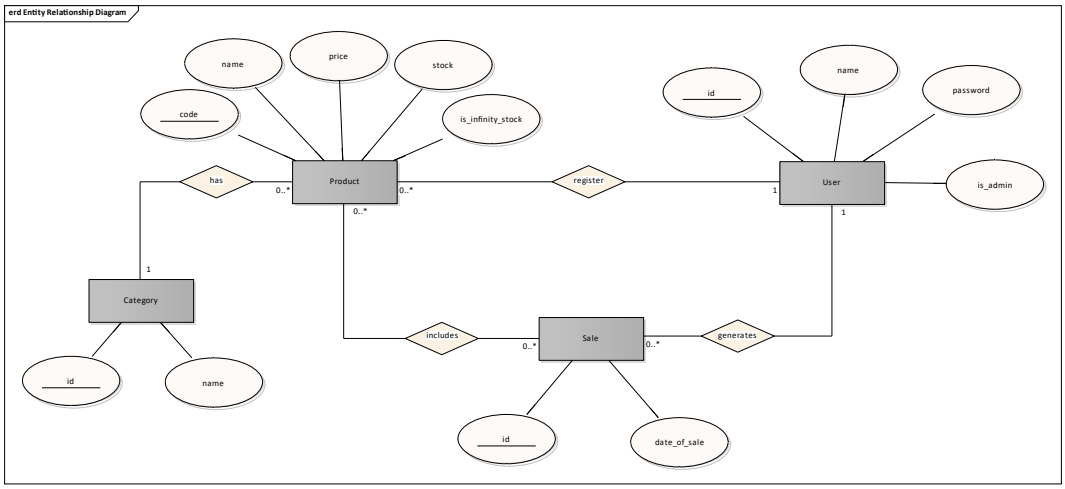
4. Diagramas UML

Los diagramas UML son producidos para presentar múltiples vistas del componente. Estos modelos proveen un model del cual se puede obtener información para ser utilizado por cualquier tipo de cliente.

4.1 Base de datos

4.1.1 Diagrama entidad-relación

Este modelo detalla las relaciones entre las entidades de la base de datos. Incluyendo como entidades Category, Product, User y Sale.



4.1.2 Conversión a tablas

A continuación se muestran las entidades en formato de tablas.

Products

code name price stock isInfinityStock category_id

Category

id name

User

id name password isAdmin

Sale

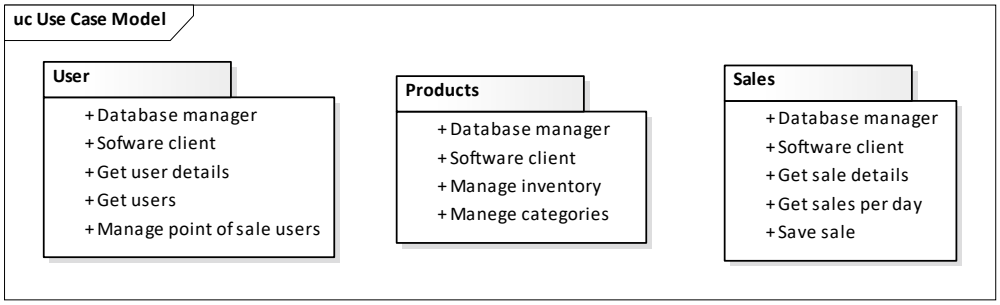
id dateOfSale id_user

Sales_products

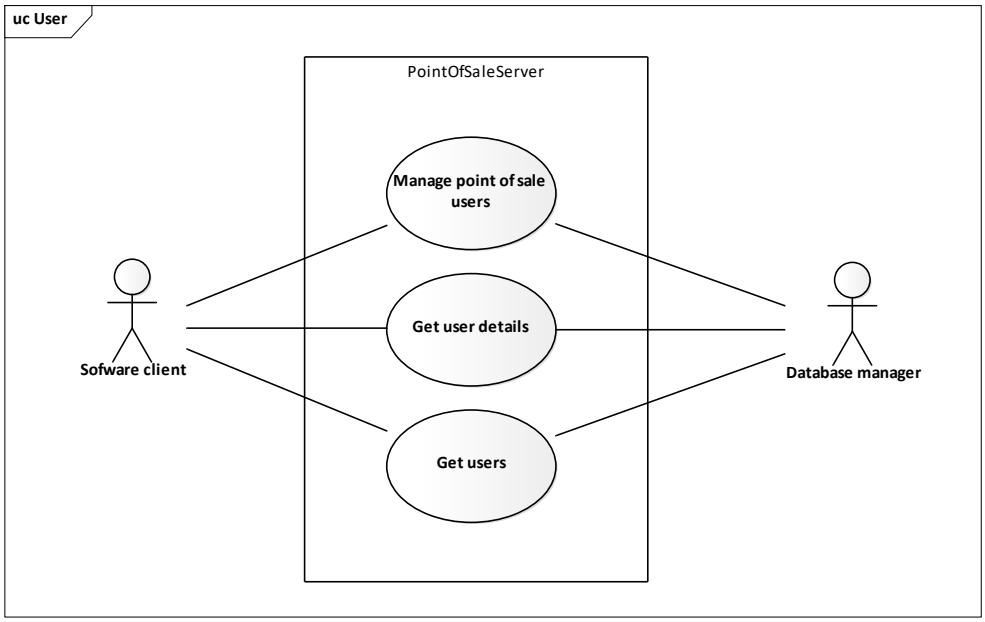
saleId productCode ProductName currentPrice units

4.2 Modelo de casos de uso

El siguiente diagrama de paquetes muestra las funcionalidades del servidor para el PointOfSale. La base de datos se maneja como un actor ya que ésta puede estar ubicada en otro servidor, por el cual, se podrá acceder a través de este componente.



4.2.1 Paquete de casos de uso - User



ID:	CU01
Nombre:	Manage point of sale users
Descripción:	Permitirá al administrador del sistema crear, actualizar o eliminar usuarios dentro de su punto de venta con el fin de mantener en orden la plantilla de trabajadores del dueño de la tienda.
Actor(es):	Software client
Precondiciones:	PRE-1: La request debe contener un token correspondiente a un USER con el rol “Admin”
Flujo Normal:	1. El server detecta una POST request por parte del software client. (ver FA2, FA3) 2. El server descompone los siguientes datos del body request: name, password y isAdmin. 3. El server valida los datos recibidos, encripta la contrasña recibida y almacena los datos a través del UserModel. (ver FA1, EX1) 4. El server agrega el status 201 al response junto con el mensaje “User created”. 5. El server manda el response al software client. 6. Fin del caso de uso.
Flujos Alternos:	FA1: Datos inválidos. 1. El server manda un response con status 400, con el mensaje “Invalid data, please verify it” 2. Regresar al paso 5 del flujo normal. FA2: *PUT request* 1. El server detecta una PUT request por parte del software client. 2. El server descompone el id desde los params del request y los siguientes datos del body request: name, password y isAdmin. 3. El server valida los datos recibidos, recupera el USER almacenado que coincida con el id recibido. (ver FA1) 4. El server verifica si a cambiado la contraseña, de ser así encripta la nueva contraseña y la establece en el USER, posteriormente almacena los nuevos datos mediante el UserModel. (ver EX1). 5. El server agrega el status 200 al response junto con el mensaje “User updated”. 6. Regresar al paso 5 del flujo normal. FA3. *DELETE request* 1. El server detecta una DELETE request por parte del software client. 2. El server descompone el id desde los params del request y recupera el USER mediante el UserModel para verificar que exista el usuario. 3. El server elimina el USER . (ver EX2) 4. El server agrega el status 200 al response junto con el mensaje “User deleted”.

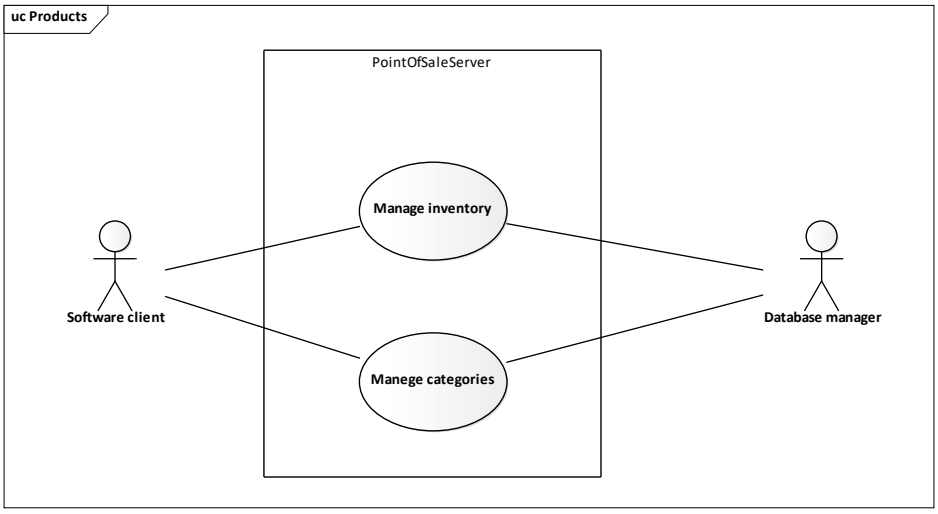
	5. Regresar al paso 5 del flujo normal.
Excepciones:	EX1: Error al guardar. 1. El server agrega al response el status 500 junto al mensaje de error. 2. Regresar al paso 5 del flujo normal. EX2: No existe un USER con el id recibido. 1. El server detecta que no existe un usuario con el id recibido. 2. El server agrega el status 500, el mensaje "User not found" al objeto JSON del response y lo manda al software client. 3. Fin del caso de uso.
Postcondiciones:	PO-1a: Se ha almacenado el usuario en la base de datos. PO-1b: Se ha actualizado el usuario en la base de datos. PO-1c: Se ha eliminado el usuario en la base de datos.
Reglas de negocio:	Ninguno.
Incluye:	Ninguno
Extiende:	Ninguno
Prioridad:	Alta

ID:	CU02
Nombre:	Get user details
Descripción:	Permitirá al administrador del sistema observar los datos de un usuario del punto de venta con el fin de decidir si necesita actualizar su información o no.
Actor(es):	Software client
Disparador:	El servidor recibe un GET *request* a la *route* "users".
Precondiciones:	PRE-1: La request debe contener un token correspondiente a un USER con el rol "Admin".
Flujo Normal:	1. El server descompone el id del request params y lo almacena en la constante id. 2. El server recupera los datos del user a través del userRepository y los almacena en un nuevo objeto USER. (ver EX1) 3. El server envía el response al software client con los siguientes datos: 1. status = 200 2. data = USER en formato JSON.
Flujos Alternos:	Ninguno.
Excepciones:	EX1: No existe un USER con el id recibido. 1. El server detecta que no existe un usuario con el id recibido. 2. El server agrega el status 400, el mensaje "User not found" al objeto JSON del response y lo manda al software client. 3. Fin del caso de uso.
Postcondiciones:	PO-1: Se ha enviado al software client el USER en formato JSON.
Reglas de negocio:	Ninguna
Incluye:	Ninguno

Extiende:	Ninguno
Prioridad:	Alta.

ID:	CU03
Nombre:	Get users
Descripción:	Permitirá al administrador del sistema observar la totalidad de usuarios registrados
Actor(es):	Software client
Disparador:	El servidor recibe un GET *request* a la *route* "user".
Precondiciones:	PRE-1: La request debe contener un token correspondiente a un USER con el rol "Admin".
Flujo Normal:	1. El server recupera todos los usuarios almacenados a través del userRepository y los almacena en la constante USERS. (ver EX1) 2. El server envía el response al software client con los siguientes datos: 1. status = 200 2. data = USERS en formato JSON ARRAY.
Flujos Alternos:	Ninguno.
Excepciones:	EX1: Error al recuperar los datos. 1. El server agrega el status 500 al response y lo manda al software client. 2. Fin del caso de uso.
Postcondiciones:	PO-1: Se ha enviado al software client los USER en formato JSON ARRAY.
Reglas de negocio:	Ninguna
Incluye:	Ninguno
Extiende:	Ninguno
Prioridad:	Alta.

4.2.2 Paquete de casos de uso – Products



ID:	CU04
Nombre:	Manage inventory
Descripción:	Permitirá al administrador del sistema y/o cajero dar de alta productos, de igual modo, permitirá modificar o eliminar la información de estos en el punto de venta con el fin de que no exista incongruencias entre la mercancía física y la registrada en la base de datos del punto de venta.
Actor(es):	Software client
Disparador:	El servidor recibe un *request* a la *route* "inventory" desde un Software client.
Precondiciones:	PRE-1: La request debe contener un token correspondiente a un USER registrado en la base de datos.
Flujo Normal:	1. El server detecta un GET request. (ver FA1, FA2, FA3, FA4) 2. El server recupera todos los PRODUCTS registrados mediante el productRepository. (ver EX1) 3. El server agrega el status 200 al response, convierte en formato JSON los PRODUCTS recuperados y los agrega al response. 4. El server envía el response al Software client. 5. Fin del caso de uso.
Flujos Alternos:	FA1: GET product details. 1. El server detecta un GET request con el productCode de parametro. 2. El server descompone el productCode del request params y lo almacena en la constante productCode. 3. El server recupera el PRODUCT correspondiente al productCode mediante el productRepository. (ver EX1, EX2) 4. El server agrega el status 200 al response, convierte en formato JSON el PRODUCT y lo agrega al response. 5. Regresar al paso 4 del flujo normal. FA2. POST request. 1. El server detecta un POST request. 2. El server descompone el code, name, price, stock, isInfinityStock y categoryId del request body y los almacena en las constantes con su respectivo nombre. 3. El server valida los datos, verifica que no exista ningún producto con el valor del code recibido, almacena los datos del PRODUCT mediante el productRepository. (ver EX2) 4. El server valida que el valor de categoryId coincida con alguna CATEGORY registrada y lo vincula al PRODUCT.(ver EX4) 5. El server agrega el status 201 al response junto con el siguiente mensaje: "Product created". 6. Regresar al paso 4 del flujo normal. FA3. PUT request 1. El server detecta un PUT request. 2. El server descompone el productCode del request params, de igual

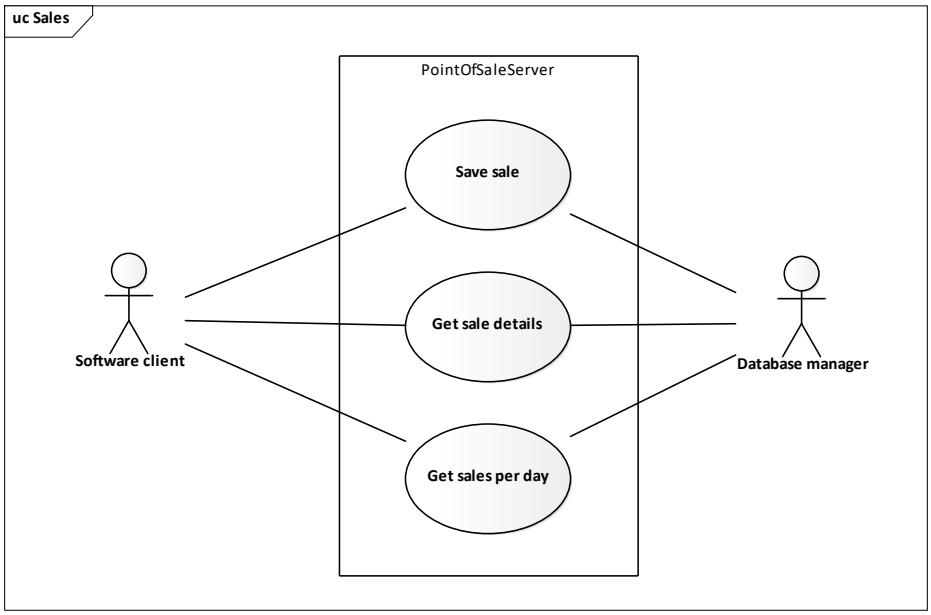
	forma, el name, price, stock, isInfinityStock y categoryId del request body y los almacena en constantes con sus respectivos nombres. 3. El server valida los datos, recupera la instancia del PRODUCT mediante el productCode, actualiza el PRODUCT mediante el productRepository. (ver EX2, ver EX3) 4. El server agrega el status 200 al response, convierte en formato JSON el PRODUCT y lo agrega al response. 5. Regresar al paso 4 del flujo normal. FA4. DELETE request. 1. El server detecta un DELETE request. 2. El server descompone el productCode del request params y lo almacena en la constante productCode. 3. El server valida que el PRODUCT exista y lo elimina de la base de datos mediante el productRepository. (ver EX3) 4. El server agrega el status 200 al response con el mensaje “Product deleted”. 5. Regresar al paso 4 del flujo normal.
Excepciones:	EX1: Error al recuperar los datos. 1. El server agrega el status 500 al response y lo manda al software client. 2. Fin del caso de uso. EX2: Datos inválidos. 1. El server manda un response con status 400, con el mensaje “Invalid data, ” más el campo correspondiente. 2. Regresar al paso 4 del flujo normal. EX3. No existe el product. 1. El server detecta que no existe el producto indicado, agrega el status 500 al response junto con el mensaje “Product not found”. 2. Regresar al paso 4 del flujo normal. EX4. No existe la categoria. 1. El server detecta que no existe ningun CATEGORY con el valor de categoryId, agrega el status 500 al response junto al mensaje “Category not found”. 2. Regresar al paso 4 del flujo normal.
Postcondiciones:	PO-1a: Se ha enviado al software client los PRODUCT en formato JSON ARRAY. PO-1b: Se ha enviado al software client el PRODUCT en formato JSONObject. PO-1c: Se ha almacena el PRODUCT y se ha enviado el response con status 201. PO-1d: Se ha actualizado la informacion del PRODUCT recibido y se ha enviado el response con status 200.

	PO-1e: Se ha eliminado la información del PRODUCT correspondiente al id recibido y se ha enviado el response con status 200.
Reglas de negocio:	Ninguna
Incluye:	Ninguno
Extiende:	Ninguno
Prioridad:	Alta.

ID:	CU05
Nombre:	Manage categories
Descripción:	Permitirá al administrador del sistema y/o cajero dar de alta, modificar o eliminar categorías con el fin de mantener organizados los productos en el inventario y poder tomar decisiones pertinentes al momento de comprar mercancía.
Actor(es):	Software client
Disparador:	El servidor recibe un *request* a la *route* "categories" desde un Software client.
Precondiciones:	PRE-1: La request debe contener un token correspondiente a un USER admin registrado en la base de datos.
Flujo Normal:	1. El server detecta un GET request. (ver FA1, FA2, FA3) 2. El server recupera todas las CATEGORY registradas mediante el categoryRepository y lo agrega al response. (ver EX1) 3. El server agrega el status 200 al response. 4. El server envía el response al software client. 5. Fin del caso de uso.
Flujos Alternos:	FA1. POST request. 1. El server detecta un POST request. 2. El server descompone el name del request body y lo almacena en una constante con el mismo nombre. 3. El server valida el name, verifica que no exista ningún CATEGORY con el valor de name y lo almacena a través del categoryRepository. (ver EX1, EX2) 4. El server agrega el status 201 al response junto con el siguiente mensaje: "Category created". 5. Regresar al paso 4 del flujo normal. FA2. PUT request. 1. El server detecta un PUT request. 2. El server descompone el id del request params y el name del request body y los almacena en constantes con el mismo nombre. 3. El server valida el name, verifica que no exista ningún CATEGORY con el valor de name y lo actualiza a través del categoryRepository. (ver EX1, EX2) 4. El server agrega el status 200 al response junto con el siguiente mensaje: "Category updated". 5. Regresar al paso 4 del flujo normal.

	FA3. DELETE request. 1. El server detecta un DELETE request. 2. El server descompone el id del request params y lo almacena en la constante id. 3. El server valida que el CATEGORY exista y lo elimina de la base de datos mediante el categoryRepository. (ver EX3) 4. El server agrega el status 200 al response con el mensaje “Category deleted”. 5. Regresar al paso 4 del flujo normal.
Excepciones:	EX1: Error al recuperar los datos. 1. El server agrega el status 500 al response y lo manda al software client. 2. Fin del caso de uso. EX2: Datos inválidos. 1. El server manda un response con status 400, con el mensaje “Invalid data, ” más el campo correspondiente. 2. Regresar al paso 4 del flujo normal. EX3. No existe la category. 1. El server detecta que no existe una CATEGORY con el valor del id recibido, agrega el status 500 al response junto con el mensaje “Category not found”. 2. Regresar al paso 4 del flujo normal.
Postcondiciones:	PO-1a: Se ha enviado al software client los CATEGORY en formato JSON ARRAY. PO-1b: Se ha almacena el CATEGORY y se ha enviado el response con status 201. PO-1c: Se ha actualizado la informacion del CATEGORY recibido y se ha enviado el response con status 200. PO-1d: Se ha eliminado la información del CATEGORY correspondiente al id recibido y se ha enviado el response con status 200.
Reglas de negocio:	Ninguna
Incluye:	Ninguno
Extiende:	Ninguno
Prioridad:	Alta.

4.2.3 Paquete de casos de uso – Sales



ID:	CU06
Nombre:	Save sale
Descripción:	Permitirá al usuario (ya sea administrador o cajero) registrar una nueva venta, para así tener almacenado cada ingreso del negocio y los ingresos sean congruentes con el dinero físico.
Actor(es):	Software client
Disparador:	El servidor recibe un POST *request* a la *route* "sales".
Precondiciones:	PRE-1: La request debe contener un token correspondiente a un USER registrado en la base de datos.
Flujo Normal:	1. El server descompone del request body el idPosUser (Identificador del usuario), dateOfSale y el array productsSold y los almacena en constantes con su mismo nombre. 2. El server valida que el valor de dateOfSale sea correcto y crea un nuevo registro SALE con la fecha recibida. (ver EX1) 3. El server recupera el USER que coincida con el valor de idPosUser y lo establece en la venta. (ver EX2) 4. El server verifica que cada producto recibido exista en la base de datos, de existir, almacena los datos de la venta de cada producto en la base de datos. (ver EX3) 5. El server agrega al response el status 201 junto al mensaje "Sale created". 6. El server envia el response al Software client. 7. Fin del caso de uso.
Flujos Alternos:	Ninguno.

Excepciones:	EX1: Fecha inválida. 1. El server detecta que el valor de dateOfSale no es correcto. 2. El server agrega el status 500 al response junto al mensaje “The date not has the YYYY-MM-DD HH:MM:SS pattern”. 3. Regresar al paso 6 del flujo normal. EX2. El usuario no existe. 1. El server detecta que no hay ningún usuario que coincida con el valor de idPosUser. 2. El server agrega el status 500 al response junto al mensaje “User not exists”. 3. Regresar al paso 6 del flujo normal. EX3. El producto no existe. 1. El server detecta que no hay ningún producto con el code recibido. 2. El server agrega el status 500 al response junto al mensaje “Product not found with ” ′+ productCode + “ code”. 3. Regresar al paso 6 del flujo normal.
Postcondiciones:	PO-1: Se ha registrado una nueva venta con todos sus productos.
Reglas de negocio:	Ninguna
Incluye:	Ninguno
Extiende:	Ninguno
Prioridad:	Alta.

ID:	CU07
Nombre:	Get sale details.
Descripción:	Permitirá al usuario (ya sea administrador o cajero) verificar los datos de una venta registrada, con el fin de validar la cantidad de dinero físico de la tienda.
Actor(es):	Software client
Disparador:	El servidor recibe un GET *request* a la *route* “sales”.
Precondiciones:	PRE-1: La request debe contener un token correspondiente a un USER registrado en la base de datos.
Flujo Normal:	1. El server descompone del request params el saleId y lo almacena en una constante del mismo nombre. 2. El server recupera el SALE que coincida con el valor de la constante saleID, además, recupera el nombre del USER que generó la venta. (ver EX1) 3. El server recupera todos los productos asociados al id del SALE y calcula el total de la venta sumando el resultado de la multiplicación de los valores de los campos currentPrice y units de cada producto. (ver EX1) 4. El server agrega el status 200 al response junto el objeto SALE descrito en la sección Variables de salida (Output variables). 5. El server envía el response al Software client.

	6. Fin del caso de uso.
Flujos Alternos:	Ninguno.
Excepciones:	EX1: Error al recuperar los datos. 1. El server detecta un error al recuperar los datos correspondientes. 2. El server agrega el status 500 al response junto al mensaje de error. 3. Regresar al paso 5 del flujo normal.
Postcondiciones:	PO-1: Se ha enviado el objeto JSON SALE al software client.
Reglas de negocio:	Ninguna
Incluye:	Ninguno
Extiende:	Ninguno
Prioridad:	Alta.

ID:	CU08
Nombre:	Get sale per day.
Descripción:	Permitirá al usuario (ya sea administrador o cajero) verificar la cantidad de ventas por un día en específico con el fin de corroborar datos físicos de productos o dinero.
Actor(es):	Software client
Disparador:	El servidor recibe un GET *request* a la *route* "sales".
Precondiciones:	PRE-1: La request debe contener un token correspondiente a un USER registrado en la base de datos.
Flujo Normal:	1. El server descompone del request params el dateToSearch y lo almacena en una constante del mismo nombre. 2. El server valida que el valor de dateToSearch tenga el patrón correcto. (ver EX1) 3. El server crea un arreglo vacío llamado SALESPERDAY. 4. El server recupera todas las SALE que en su campo dateOfSale contenga el valor de dateToSearch. (ver EX2) 5. El server recupera todos los productos asociados al id de cada SALE y calcula el total de cada venta sumando el resultado de la multiplicación de los valores de los campos currentPrice y units de cada producto de ésta. (ver EX2) 6. El server agrega cada SALE al arreglo SALESPERDAY. 7. El server agrega el status 200 al response junto el arreglo SALESPERDAY descrito en la sección Variables de salida (Output variables). 8. El server envía el response al Software client. 9. Fin del caso de uso.
Flujos Alternos:	Ninguno.
Excepciones:	EX1. Fecha invalida. 1. El server detecta que el valor de dateToSearch no coincide con el patrón establecido. 2. El server agrega el status 500 al response junto al mensaje "The date

	not has the YYYY-MM-DD pattern". 3. Regresar al paso 8 del flujo normal. EX2: Error al recuperar los datos. 1. El server detecta un error al recuperar los datos correspondientes. 2. El server agrega el status 500 al response junto al mensaje de error. 3. Regresar al paso 5 del flujo normal.
Postcondiciones:	PO-1: Se ha enviado el array JSON SALESPERDAY al software client.
Reglas de negocio:	Ninguna
Incluye:	Ninguno
Extiende:	Ninguno
Prioridad:	Alta.

5. Modelo de componentes (Component Model)

Mediante el siguiente modelo de componentes se pretende mostrar como cada módulo del proyecto (Users, Categories, Products, Sales y Authentication) exponen sus métodos para ser utilizados por el Software client o entre ellos mismos.

Las interfaces expuestas del PointOfSaleServer se exponen como métodos que tienen como parámetros un request (req) y un response (res), debido a que se utiliza el framework Express (paquete de Node.js)

Todos los métodos utilizan la interfaz expuesta verifyToken del módulo Authentication, pero para no entorpecer la visibilidad y legibilidad del modelo, se omitieron esas conexiones.