UNIVERSIDAD CATÓLICA DEL MAULE PROFESOR
Ingeniería Ejecución en Computación e Informática Mg. Jerman Espíndola Sandoval

PROGRAMACIÓN BACK END

EVALUACIÓN Nº 3

Resultados de Aprendizaje a evaluar en este instrumento:

- Construir una API REST utilizando framework del lado del servidor, conforme a requerimientos de un proyecto informático dentro de un contexto regional real o simulado, implementando conexión a base de datos.

Criterios de evaluación de este instrumento:

- Configura conexión a base de datos.
- Codifica funciones que realicen operaciones CRUD sobre una base de datos conforme a requerimientos del proyecto informático de los estudiantes.
- Codifica instrucciones que generan salidas en formato JSON.
- Da solución a un problema dentro de un contexto regional real o simulado.

Descripción del instrumento:

Este instrumento se compone de 1 ejercicio de desarrollo extenso.

Puntaje Ideal**: 150 pts.** Puntaje para nota 4.0 (60%) = **89 pts.**

Considere la siguiente situación de desempeño

tiendaenlinea.cl es un portal online que vende todo tipo de productos de retail (ropa, calzado, electrodomésticos, entre otros) para el mercado chileno.

Actualmente la empresa cuenta con una aplicación monolítica construida en el año 2010 utilizando el siguiente stack tecnológico.

Capa de presentación (vista)	Java Server Pages (JSP) HTML 4 Javascript

	CSS
Capa de negocios (controlador)	Servlets 3.0
Capa de datos (modelo)	EJB 3.0 JPA 1.0
Base de datos	MySQL 5

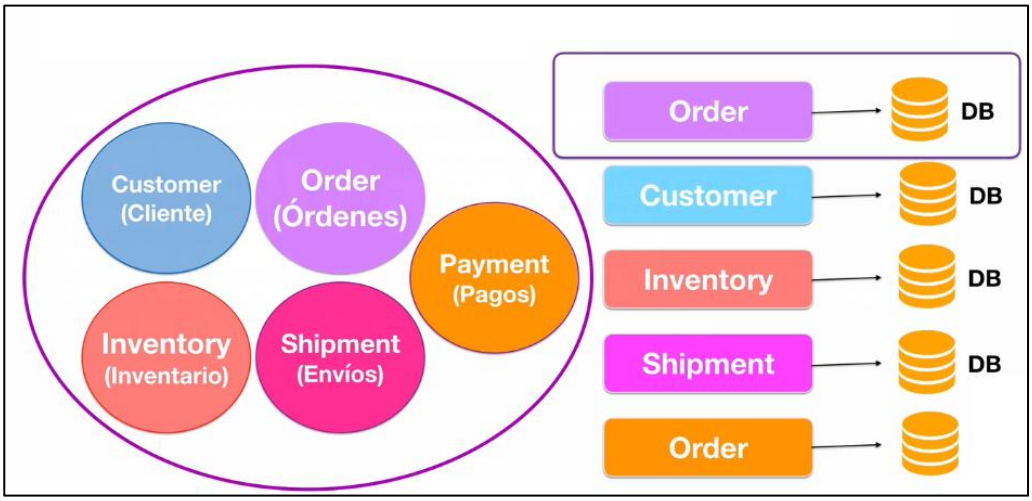
La empresa pretende expandirse al resto de los países de Latinoamérica, para esto desean que su aplicación sea altamente escalable y resiliente, con el fin de soportar las cientos de peticiones diarias actuales, más las que provendrán de los nuevos mercados.

tiendaenlinea.cl tiene grandes ambiciones de expansión, y apuesta a contar con un stack tecnológico que le permita realizar cambios de manera flexible y crecer orgánicamente. Por lo tanto, los microservicios son la tecnología seleccionada.

El nuevo stack tecnológico es el siguiente:

Capa de presentación (vista)	ReactJS HTML 5 Javascript CSS 3
Capa de negocios (controlador)	Rest API (Django Rest Framework)
Capa de datos (modelo)	Models (Django Rest Framework)
Base de datos	MySQL 8 o superior

La nueva arquitectura de la capa de negocios (basada en microservicios) es la siguiente:



I. **Implemente el microservicio Customer (Cliente) mediante la creación de un API Rest con integración a base de datos.**

Considere la siguiente estructura para el modelo de datos

Debido a que en paralelo otro equipo de desarrollo construirá la interfaz gráfica de la aplicación, es necesario que el microservicio *Customer* se acoja a los siguientes Contratos Rest:

Método HTTP	**URL**	**Descripción**
GET	customers/	Retorna todos los clientes, si la petición es exitosa, envía un código de estado **HTTP 200 (OK)**.
POST	customer/	Crea un nuevo cliente, la solicitud debe incluir los datos del nuevo cliente dentro del cuerpo (body). Si el cliente es creado retorna los datos del nuevo cliente y envía un código de estado **HTTP 201 (Created)**.
GET	customer/<id>	Retorna un cliente basado en su *id* asociado, si el cliente no existe, envía un código de

		estado **HTTP 404 (Not Found)**, en caso de ser encontrado, retorna el cliente y envía un código de estado **HTTP 200 (OK)**.
PUT	customer/<id>	Actualiza un cliente existente, si el cliente no puede ser encontrado, envía un código de estado **HTTP 404 (Not Found)**, en caso de ser encontrado y actualizado, retorna el cliente actualizado y envía un código de estado **HTTP 200 (OK)**.
DELETE	customer/<id>	Elimina un cliente basado en su *id* asociado, si el cliente no existe, envía un código de estado **HTTP 404 (Not Found)**, en caso de ser encontrado y eliminado, envía un código de estado **HTTP 204 (Not Content)**.

Método HTTP	**URL**	**Descripción**
GET	address/<customer_id>	Retorna todas las direcciones de un cliente basado en su *id* asociado, si el cliente no tiene direcciones registradas, envía un código de estado **HTTP 404 (Not Found)**, en caso contrario, retorna las direcciones registradas y envía un código de estado **HTTP 200 (OK)**.
POST	address/	Crea una nueva dirección, la solicitud debe incluir los datos de la nueva dirección dentro del cuerpo (body). Si la dirección es creada retorna los datos de la

		nueva dirección y envía un código de estado **HTTP 201 (Created)**.
PUT	address/<id>	Actualiza una dirección existente basado únicamente en su *id* asociado (no en el *id* del cliente), si la dirección no puede ser encontrada, envía un código de estado **HTTP 404 (Not Found)**, en caso de ser encontrada y actualizada, retorna la dirección actualizada y envía un código de estado **HTTP 200 (OK)**.
DELETE	address/<id>	Elimina una dirección existente basado únicamente en su *id* asociado (no en el *id* del cliente), si la dirección no existe, envía un código de estado **HTTP 404 (Not Found)**, en caso de ser encontrada y eliminada, envía un código de estado **HTTP 204 (Not Content)**.

Consideraciones:

- Debe aplicar vistas basadas en función.
- Debe aplicar vistas basadas en clases.
- Implementar autenticación basada en JWT (JSON Web Tokens).
    - Utilizar django-rest-framework-simplejwt
    - Los tokens deben expirar en 5 minutos.

- Implementar, como mínimo, dos MIXINS.
- Implementar, como mínimo, dos GENERICS.
- Implementar, como mínimo, un VIEW SETS.
- El estándar de transferencia de datos a utilizar es JSON.

Al finalizar comprima la carpeta del proyecto y suba el archivo a la actividad "Evaluación N° 3", dentro de LMS.