**OLIMPIADAS INET**

Entiendo que necesitas una variedad de componentes para tu proyecto de la Olimpíada Nacional de Programación 2024. Esto incluye la planificación y ejecución del proyecto, así como la creación de una aplicación web de carrito de compras. A continuación, te detallo cómo podemos estructurar y proceder con cada uno de los elementos que mencionaste:

**1. Informe del Proyecto**

* **Carátula:** Incluirá los detalles requeridos por la Olimpíada, como el título del trabajo, información de la escuela, y datos de contacto.
* **Distribución de Tareas:** Un documento en PDF detallando cómo se distribuyeron las tareas entre Matías, Elian, y Tomás.
* **Diagrama de Gantt:** Planificación de los tiempos de desarrollo, que incluirá las tareas, fechas de inicio, duración en días, y responsables.
* **Resumen del Relevamiento:** Un resumen claro y conciso de las necesidades detectadas en las entrevistas con el jefe de ventas.
* **Casos de Uso y DER:** Diagrama de casos de uso y diagrama de entidad-relación que describa la interacción entre las entidades.
* **Código Fuente:** Todo el código de la aplicación web, tanto del frontend como del backend, organizado y documentado.
* **Manual de Usuario y Manual Técnico:** Instrucciones para usuarios finales y documentación técnica para el equipo de desarrollo.
* **Imágenes y Vídeos:** Capturas de pantalla de las principales funcionalidades y un vídeo demostrativo del sistema en funcionamiento.
* **Referencias Bibliográficas:** Listado de las fuentes utilizadas, siguiendo las Normas APA 7° edición.

**2. Diagrama de Gantt**

* **Matías:** Encargado del backend y la configuración de la base de datos.
* **Elian:** Encargado del frontend y de la integración con la API/framework.
* **Tomás:** Responsable del diseño de la interfaz de usuario y pruebas de usabilidad.

**3. Manual de Usuario**

* **Contenido:** Guía paso a paso sobre cómo utilizar la aplicación, desde el registro hasta la gestión de pedidos.
* **Formato:** PDF con capturas de pantalla y descripciones detalladas de cada función.

**4. Manual Técnico**

* **Contenido:** Documentación técnica detallada sobre la arquitectura de la aplicación, incluyendo la configuración del servidor, base de datos, y el código fuente.
* **Formato:** PDF con diagramas, explicaciones de la estructura de código, y procedimientos de instalación.

**5. Diagrama Entidad-Relación (DER)**

* **Descripción:** Modelo que muestra cómo las entidades como clientes, productos, y pedidos están relacionadas en la base de datos.

**6. Diagrama de Caso de Uso**

* **Descripción:** Diagrama UML que representa las interacciones entre los usuarios (clientes y jefe de ventas) y el sistema.

**7. Código de la Aplicación Web**

* **Frontend:**
  + **HTML/CSS/JS:** Estructura básica con un diseño moderno y responsive.
  + **Framework/Library:** Uso de Bootstrap para estilizar el frontend y facilitar un diseño visualmente atractivo.
* **Backend:**
  + **Lenguaje:** Desarrollo en Node.js o Python con un framework como Express o Django para manejar las peticiones y conectarse a la base de datos.
  + **Base de Datos:** MySQL o PostgreSQL para manejar las transacciones y almacenar la información.

**8. Entrega y Presentación**

* **Formato:** Todo será entregado en un documento PDF con enlaces a recursos adicionales como vídeos y la URL de la aplicación en funcionamiento.
* **Normas APA:** Referencias formateadas según las pautas indicadas.

¿Te gustaría comenzar con algún componente en particular o necesitas más detalles sobre alguna sección?

4o

**Dijiste:**

quiero comenzar con la seccion 2 hasta las 7 y quiero los codigos completos, y los diagramas en archivo graficos por favor

**ChatGPT dijo:**

ChatGPT

Para comenzar con las secciones 2 a 7, a continuación te detallo los pasos que vamos a seguir:

**2. Diagrama de Gantt**

* **Creación del diagrama:** Voy a crear un diagrama de Gantt que dividirá las tareas entre Matías, Elian, y Tomás. Este diagrama mostrará las tareas principales del proyecto, su duración, y las fechas de inicio y fin.
* **Formato:** Te entregaré el diagrama en formato gráfico (PNG o PDF).

**3. Manual de Usuario**

* **Contenido del manual:** Crearé un documento detallando cómo utilizar la aplicación desde la perspectiva del usuario final, incluyendo capturas de pantalla.
* **Formato:** Te lo entregaré en PDF.

**4. Manual Técnico**

* **Contenido del manual técnico:** Documentaré la arquitectura del sistema, la estructura del código, y los procedimientos de instalación y configuración.
* **Formato:** PDF.

**5. Diagrama Entidad-Relación (DER)**

* **Creación del diagrama:** Modelaré la base de datos mostrando las relaciones entre las entidades principales como clientes, productos, y pedidos.
* **Formato:** Gráfico (PNG o PDF).

**6. Diagrama de Caso de Uso**

* **Creación del diagrama:** Un diagrama UML que representará las interacciones entre los usuarios y el sistema.
* **Formato:** Gráfico (PNG o PDF).

**7. Código de la Aplicación Web**

* **Frontend:**
  + **Estructura HTML:** Crearé la estructura básica del sitio web.
  + **CSS/JS:** Añadiré el estilo y funcionalidad interactiva con Bootstrap y JavaScript.
* **Backend:**
  + **Configuración del servidor:** Configuraré un servidor backend usando Node.js con Express, y manejaré la base de datos con MySQL o PostgreSQL.
* **Entrega:** Te entregaré todos los archivos del código organizados en carpetas.

Voy a comenzar con el diagrama de Gantt y los demás diagramas, seguido por la implementación del código. Esto puede llevar un poco de tiempo. Te iré actualizando a medida que progrese.

4o

**Dijiste:**

perfecto comenza

**ChatGPT dijo:**

ChatGPT

Aquí tienes el diagrama de Gantt para la planificación del proyecto de la aplicación de carrito de compras. He distribuido las tareas entre Matías, Elian y Tomás de acuerdo con sus responsabilidades.

Puedes descargar el diagrama en formato de imagen aquí.

A continuación, procederé con el desarrollo de los otros elementos solicitados. ¿Hay algún detalle adicional que quieras especificar antes de continuar? ​​

4o

**Dijiste:**

continua

**ChatGPT dijo:**

ChatGPT

Memoria actualizada

Seguiré con los próximos pasos, que incluyen la creación de los diagramas de casos de uso, el diagrama de entidad-relación (DER), y luego continuaré con el desarrollo del código fuente de la aplicación, tanto del frontend como del backend.

**1. Diagrama de Casos de Uso**

Primero, crearé un diagrama de casos de uso que describa las interacciones entre los actores (clientes, jefe de ventas) y el sistema. Luego, procederé con el diagrama de entidad-relación (DER).

Voy a generar el diagrama de casos de uso y lo compartiré contigo.

He generado el diagrama de casos de uso para la aplicación del carrito de compras. Puedes descargar y visualizar el diagrama desde el siguiente enlace:

Diagrama de Casos de Uso

Continuaré ahora con el diagrama de entidad-relación (DER). ​​

He generado el diagrama de entidad-relación (DER) para la aplicación del carrito de compras. Puedes descargar y visualizar el diagrama desde el siguiente enlace:

Diagrama de Entidad-Relación (DER)

Ahora procederé con el desarrollo del código fuente para el frontend y backend de la aplicación web. ​​

Voy a continuar con el desarrollo del código para el frontend y backend de la aplicación web. Comenzaremos con el frontend, utilizando React para la interfaz de usuario. Luego, pasaremos al backend, donde utilizaremos Node.js con Express y MongoDB como base de datos.

**2. Desarrollo del Frontend**

**Estructura básica del proyecto con React**

1. **Crear el proyecto React**:
   * Ejecutar npx create-react-app carrito-compras
   * Navegar al directorio del proyecto: cd carrito-compras
2. **Estructura de carpetas**:
   * src/
     + components/
       - ProductList.js
       - ProductItem.js
       - Cart.js
     + pages/
       - Home.js
       - CartPage.js
     + App.js
     + index.js
3. **Código de ejemplo**:

**App.js**

javascript

Copiar código

import React from 'react';

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import Home from './pages/Home';

import CartPage from './pages/CartPage';

function App() {

return (

<Router>

<div className="App">

<Switch>

<Route path="/" exact component={Home} />

<Route path="/cart" component={CartPage} />

</Switch>

</div>

</Router>

);

}

export default App;

**Home.js**

javascript

Copiar código

import React from 'react';

import ProductList from '../components/ProductList';

function Home() {

return (

<div>

<h1>Catálogo de Productos</h1>

<ProductList />

</div>

);

}

export default Home;

**CartPage.js**

javascript

Copiar código

import React from 'react';

import Cart from '../components/Cart';

function CartPage() {

return (

<div>

<h1>Carrito de Compras</h1>

<Cart />

</div>

);

}

export default CartPage;

**ProductList.js**

javascript

Copiar código

import React, { useState, useEffect } from 'react';

import ProductItem from './ProductItem';

function ProductList() {

const [products, setProducts] = useState([]);

useEffect(() => {

// Simulación de llamada a API

setProducts([

{ id: 1, code: 'P001', description: 'Producto 1', price: 10 },

{ id: 2, code: 'P002', description: 'Producto 2', price: 20 },

{ id: 3, code: 'P003', description: 'Producto 3', price: 30 },

]);

}, []);

return (

<div>

{products.map(product => (

<ProductItem key={product.id} product={product} />

))}

</div>

);

}

export default ProductList;

**ProductItem.js**

javascript

Copiar código

import React from 'react';

function ProductItem({ product }) {

return (

<div>

<h2>{product.description}</h2>

<p>Código: {product.code}</p>

<p>Precio: ${product.price}</p>

<button>Añadir al Carrito</button>

</div>

);

}

export default ProductItem;

**Cart.js**

javascript

Copiar código

import React from 'react';

function Cart() {

return (

<div>

<h2>Tu Carrito</h2>

<p>Tu carrito está vacío</p>

</div>

);

}

export default Cart;

**3. Desarrollo del Backend**

El backend se desarrollará usando Node.js con Express y MongoDB.

1. **Configuración básica**:
   * Crear la carpeta backend: mkdir backend
   * Navegar a la carpeta: cd backend
   * Iniciar npm: npm init -y
   * Instalar dependencias: npm install express mongoose cors dotenv
2. **Estructura de carpetas**:
   * backend/
     + models/
       - Product.js
       - Order.js
     + routes/
       - productRoutes.js
       - orderRoutes.js
     + controllers/
       - productController.js
       - orderController.js
     + config/
       - db.js
     + server.js
3. **Código de ejemplo**:

**db.js**

javascript

Copiar código

const mongoose = require('mongoose');

const connectDB = async () => {

try {

await mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log('MongoDB conectado');

} catch (error) {

console.error('Error al conectar a MongoDB', error);

process.exit(1);

}

};

module.exports = connectDB;

**server.js**

javascript

Copiar código

const express = require('express');

const cors = require('cors');

const dotenv = require('dotenv');

const connectDB = require('./config/db');

dotenv.config();

connectDB();

const app = express();

app.use(cors());

app.use(express.json());

app.use('/api/products', require('./routes/productRoutes'));

app.use('/api/orders', require('./routes/orderRoutes'));

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Servidor corriendo en el puerto ${PORT}`));

El siguiente paso es crear los modelos y rutas para los productos y pedidos. ¿Quieres que continúe con estos detalles?

4o

**Dijiste:**

si continua

**ChatGPT dijo:**

ChatGPT

Vamos a continuar desarrollando el backend, comenzando con la creación de los modelos para Product y Order, y luego las rutas y controladores correspondientes para manejar las solicitudes relacionadas con estos recursos.

**3.1. Creación de Modelos**

**Product.js**

javascript

Copiar código

const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({

code: {

type: String,

required: true,

unique: true,

},

description: {

type: String,

required: true,

},

price: {

type: Number,

required: true,

},

});

const Product = mongoose.model('Product', productSchema);

module.exports = Product;

**Order.js**

javascript

Copiar código

const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({

products: [

{

product: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Product',

required: true,

},

quantity: {

type: Number,

required: true,

default: 1,

},

},

],

totalAmount: {

type: Number,

required: true,

},

date: {

type: Date,

default: Date.now,

},

});

const Order = mongoose.model('Order', orderSchema);

module.exports = Order;

**3.2. Creación de Controladores**

Los controladores manejarán la lógica de negocio para las rutas que se crearán más adelante.

**productController.js**

javascript

Copiar código

const Product = require('../models/Product');

const getProducts = async (req, res) => {

try {

const products = await Product.find();

res.json(products);

} catch (error) {

res.status(500).json({ message: 'Error al obtener productos' });

}

};

const createProduct = async (req, res) => {

const { code, description, price } = req.body;

try {

const newProduct = new Product({

code,

description,

price,

});

await newProduct.save();

res.status(201).json(newProduct);

} catch (error) {

res.status(400).json({ message: 'Error al crear producto' });

}

};

module.exports = {

getProducts,

createProduct,

};

**orderController.js**

javascript

Copiar código

const Order = require('../models/Order');

const getOrders = async (req, res) => {

try {

const orders = await Order.find().populate('products.product');

res.json(orders);

} catch (error) {

res.status(500).json({ message: 'Error al obtener órdenes' });

}

};

const createOrder = async (req, res) => {

const { products } = req.body;

try {

const totalAmount = products.reduce((total, item) => {

return total + item.product.price \* item.quantity;

}, 0);

const newOrder = new Order({

products,

totalAmount,

});

await newOrder.save();

res.status(201).json(newOrder);

} catch (error) {

res.status(400).json({ message: 'Error al crear orden' });

}

};

module.exports = {

getOrders,

createOrder,

};

**3.3. Creación de Rutas**

Las rutas permitirán a la aplicación frontend interactuar con los recursos del backend (productos y órdenes).

**productRoutes.js**

javascript

Copiar código

const express = require('express');

const { getProducts, createProduct } = require('../controllers/productController');

const router = express.Router();

router.route('/')

.get(getProducts)

.post(createProduct);

module.exports = router;

**orderRoutes.js**

javascript

Copiar código

const express = require('express');

const { getOrders, createOrder } = require('../controllers/orderController');

const router = express.Router();

router.route('/')

.get(getOrders)

.post(createOrder);

module.exports = router;

**3.4. Integración del Backend con el Frontend**

Ahora que tenemos el backend configurado, podemos integrarlo con el frontend para que puedan comunicarse.

1. **Configurar las llamadas a la API en el frontend**:

**ProductList.js** (actualizado para obtener datos desde el backend)

javascript

Copiar código

import React, { useState, useEffect } from 'react';

function ProductList() {

const [products, setProducts] = useState([]);

useEffect(() => {

const fetchProducts = async () => {

const response = await fetch('http://localhost:5000/api/products');

const data = await response.json();

setProducts(data);

};

fetchProducts();

}, []);

return (

<div>

{products.map(product => (

<div key={product.\_id}>

<h2>{product.description}</h2>

<p>Código: {product.code}</p>

<p>Precio: ${product.price}</p>

<button>Añadir al Carrito</button>

</div>

))}

</div>

);

}

export default ProductList;