Universidad Rafael Landívar  
Campus Central  
Dirección de Tecnologías de la Información

Prueba Técnica para Analista Desarrollador II

Realizada por: Carlos Daniel Barrientos Castillo

Nueva Guatemala de la Asunción, agosto 24 del 2025

Tabla de contenido

[Enfoque del Documento: 3](#_Toc206926301)

[Requerimientos: 4](#_Toc206926302)

[Enfoque del Producto: 5](#_Toc206926303)

[Giro del Negocio: 6](#_Toc206926304)

[Arquitectura Utilizada: 7](#_Toc206926305)

[Diagramas de Conceptos: 8](#_Toc206926306)

[Interconexion de Componentes: 9](#_Toc206926307)

[Instalación para Desarrolladores: 10](#_Toc206926308)

[Recomendaciones: 11](#_Toc206926309)

[Referencias 12](#_Toc206926310)

# Enfoque del Documento:

Dentro del siguiente documento se defiende la arquitectura utilizada para la prueba técnica para el puesto de Analista Desarrollador II en el Departamento de Tecnologías de la Información, el nombre del proyecto fue nombrado como “Foodie Test”, por lo cual dentro del proyecto y siguiente documento se encontrarán multiplex referencias a este nombre para recalcarlo como el nombre del proyecto.

En el siguiente documento se podrá encontrar todo lo necesario para la evaluación del proyecto desde una perspectiva más técnica y con un enfoque mas practico, por lo que se recomienda que se tengan los siguientes conceptos claros.

* API
* Json Web Token
* React
* Axios
* Servicios Web
* Bases de datos relacionales
* Node.js
* Express

# Requerimientos:

Dentro del documento origen de la prueba técnica se especifican los siguientes requerimientos técnicos funcionales y no explictos. Por lo que el proyecto se limita a estos requerimientos

## Requerimientos Funcionales:

* Registro de clientes
* Registro de compras
* Visibilidad Restringida para clientes
* Gestiones por administradores
* Envió de Ofertas o Anuncios
* Eliminación de Clientes
* Edición de datos básicos del cliente
* Interfaz grafica conectada a una API o servicio Web
* Autenticación de usuarios
* Autorización por perfiles
* CRUD de clientes parcial (Crear, leer, editar y eliminar, pero solo por administradores)
* CRUD de compras parcial (Crear, leer por cliente y lectura por administradores)

## Requerimientos no explícitos:

* Almacenamiento seguro de credenciales (hash de contraseñas)
* Controles de autorización en API
* Privacidad por parte de los datos entre clientes
* Rendimiento de la API
* Disponibilidad y confiabilidad
* Manejo de errores con la interfaz grafica
* Usabilidad
* Pruebas básicas
* Portabilidad

## Criterios de Aceptación:

* Un cliente puede registrarse, iniciar sesión, cerrar sesión y ver sus propias compras
* Un administrador puede iniciar sesión, cerrar sesión, ver el listado de clientes y sus detalles, editar clientes, eliminar usuarios y enviar ofertas/anuncios
* La UI consume la API, ambos entregables vienen con instrucciones
* El repositorio debe ser público, con una rama master y el commit final debe ser previo al 24/08/2025 a las 20:00

# Enfoque del Producto:

El enfoque del producto es crear una aplicación web que pueda ayudar a una empresa ficticia a conectar a los usuarios con el negocio, esto por medio de envió de ofertas y anuncios a los usuarios por medio de la misma aplicación web. Los usuarios de la aplicación pueden comprar productos desde la misma aplicación desde un dashboard creado solo para ellos que lleva el registro de sus compras y anuncios.

Los administradores pueden realizar diferentes actividades por medio de un dashboard especial con los usuarios como lo seria editar la información de cada uno, eliminar a los usuarios del sistema y enviar las ofertas y anuncios

# Arquitectura Utilizada:

Para la arquitectura se eligió la manera más rápida de completar el producto en el tiempo estimado, para lo cual fue necesario utilizar React para el frontend, utilizando Bootstrap y Node.js con Express para el backend, para la base de datos relacional se utilizó sqlite. A continuación, se presentan los puntos de la elección de las tecnologías.

### React + Bootstrap:

Agilidad, se requiere de un proyecto que pueda estar listo en menos de 12 horas, por lo cual utilizar React que ya tiene sus propias maneras de distribuir components y separar las responsabilidades hizo más rápido el trabajo de desarrollo, en cuanto a Bootstrap, requeríamos una librería confiable que pudiera adaptarse a cualquier proyecto sin necesidad de definir muchos estilos y tener combinaciones por defecto para centrarse en la funcionalidad

### Node.js + Express:

Confiabilidad, se necesitó una plataforma segura con lo cual el tener JWT para separar responsabilidades y sesiones de usuario que era algo indispensable para el proyecto, por lo cual el tener una plataforma con buena compatibilidad y fácil adoptabilidad era muy necesario, node.js por su parte nos ofrece una buena adaptabilidad entre con los JWT aparte de que crear una API con ayuda de Express nos daba mas facilidad y agilidad para no perder tiempo en configuraciones como lo hubiéramos tenido que hacer en .NET.

# Diagramas de Conceptos:

# Interconexión de Componentes:

La API del backend cuenta con los siguientes endpoints:

Autenticación:

* POST:/auth/register:
  + Requiere 3 parámetros: nombre, correo y clave del usuario
  + Inserta un registro a la tabla Clientes
* POST: /auth/login
  + Requiere 2 parámetros: correo y clave del usuario
  + Realiza una selección del usuario que tenga el mismo correo que se mando en el parámetro, se le aplica un hash a la contraseña del parámetro y si coinciden las contraseñas, regresa el token, el rol del usuario y establece una sesión de 2 horas

Clientes:

* GET: /clients/:
  + Requiere 3 parámetros: nombre, correo y clave del usuario
  + Inserta un registro a la tabla Clientes
* GET: /clients/me
  + Requiere 2 parámetros: correo y clave del usuario
  + Realiza una selección del usuario que tenga el mismo correo que se mando en el parámetro, se le aplica un hash a la contraseña del parámetro y si coinciden las contraseñas, regresa el token, el rol del usuario y establece una sesión de 2 horas
* GET: /clients/delete:
  + Requiere 3 parámetros: nombre, correo y clave del usuario
  + Inserta un registro a la tabla Clientes

Ofertas:

* POST: /offers/:
  + Requiere 3 parámetros: nombre, correo y clave del usuario
  + Inserta un registro a la tabla Clientes
* GET: /offers/me/
  + Requiere 2 parámetros: correo y clave del usuario
  + Realiza una selección del usuario que tenga el mismo correo que se mando en el parámetro, se le aplica un hash a la contraseña del parámetro y si coinciden las contraseñas, regresa el token, el rol del usuario y establece una sesión de 2 horas

Compras:

* POST:/purchases/:
  + Requiere 3 parámetros: nombre, correo y clave del usuario
  + Inserta un registro a la tabla Clientes
* GET:/purchases/:
  + Requiere 3 parámetros: nombre, correo y clave del usuario
  + Inserta un registro a la tabla Clientes
* GET:/purchases /me
  + Requiere 2 parámetros: correo y clave del usuario
  + Realiza una selección del usuario que tenga el mismo correo que se mando en el parámetro, se le aplica un hash a la contraseña del parámetro y si coinciden las contraseñas, regresa el token, el rol del usuario y establece una sesión de 2 horas

# Instalación para Desarrolladores:

# Recomendaciones:

# Referencias

**No hay ninguna fuente en el documento actual.**