## Control del Documento

**Proyecto**

*OptiService*

**Título**

Arquitectura del Sistema

**Historial de versiones**

|  |  |  |
| --- | --- | --- |
| **Titulo** | **Versión** | **Fecha** |
| Arquitectura del sistema | 1.0 | 11 de septiembre de 2023 |
| Vista lógica | 2.0 | 14 de noviembre de 2023 |
| Vista componentes | 3.0 | 20 de noviembre de 2023 |
| Instalación de paquetes | 4.0 | 10 de diciembre de 2023 |
| Organizacion | 5.0 | 9 de febrero de 2024 |

**Generado por**

Grupo de Ingeniería: *Alex Pazmiño y Anderson Santana*

**Aprobado por**

*Dr. Rubén Pazmiño*

**Alcance de la distribución**

*Documento Interno – alcance a toda los stakeholders.*

## Índice

[Control del Documento 1](#_TOC_250032)

[Índice 2](#_TOC_250031)

[SOBRE ESTE DOCUMENTO 4](#_TOC_250030)

[DESCRIPCIÓN GENERAL 5](#_TOC_250029)

[Posicionamiento y Alcance 5](#_TOC_250028)

[Objetivos del diseño 5](#_TOC_250027)

[Características Principales 5](#_TOC_250026)

[Restricciones y limitaciones 5](#_TOC_250025)

[REQUISITOS ATENDIDOS 6](#_TOC_250024)

[Requisitos Funcionales 6](#_TOC_250023)

[Requisitos No Funcionales 6](#_TOC_250022)

[VISTAS, CAPAS Y PLANOS 7](#_TOC_250021)

[Resumen de Capas 7](#_TOC_250020)

[Vista Lógica 7](#_TOC_250019)

[Vista de Componentes 7](#_TOC_250018)

[Vista de Despliegue 8](#_TOC_250017)

[Vista de Datos 8](#_TOC_250016)

[DETALLES DE IMPLEMENTACIÓN 9](#_TOC_250015)

[Plataformas y Lenguajes 9](#_TOC_250014)

[Organización del proyecto 10](#_TOC_250013)

Obtención e instalación del paquete de fuentes 10

[Contenido del paquete de fuentes 10](#_TOC_250012)

[Instrucciones de compilación 10](#_TOC_250011)

[Estructuras de implementación 10](#_TOC_250010)

[ANEXOS 11](#_TOC_250009)

[Título del primer anexo 11](#_TOC_250008)

[Historial del Documento 11](#_TOC_250007)

[Referencias a otros documentos 11](#_TOC_250006)

[Insumos 11](#_TOC_250005)

[Documentos derivados 12](#_TOC_250004)

[Glosario de términos 12](#_TOC_250003)

[Significado de los elementos de la notación gráfica 12](#_TOC_250002)

[Estereotipado UML utilizado 12](#_TOC_250001)

[Significado de los elementos No UML 12](#_TOC_250000)

# Sobre este Documento

El presente documento contiene el diseño elaborado para el proyecto ***OptiService***, el cual es producto de un análisis minucioso de los requisitos del sistema, según estos pueden ser satisfechos con las tecnologías y características discutidas con los clientes y usuarios.

El documento está organizado alrededor de tres ideas principales:

1. Las características generales del diseño
2. Los requisitos atendidos por el diseño
3. Los modelos y vistas que lo detallan

Al contrario de muchas otras actividades técnicas, el desarrollo de sistemas intensivos en software dedica la mayoría de sus esfuerzos a la especificación y modelado.

Los modelos son utilizados tanto para el análisis de requisitos, como para el diseño de la solución, así como para la especificación, construcción y despliegue del sistema en su ambiente de explotación.

Por otro lado, los programas de computadora son construidos por medio del uso de herramientas de traducción automáticas llamados *compiladores*[*1*](#_bookmark0), para los cuales es construida la forma final y más detallada del software del sistema: *el código fuente*.

La última sección del documento indica la forma en que se puede obtener el código fuente del proyecto, así como las instrucciones de compilación necesarias para lograr la ejecución de los componentes que este código detalla.

# Descripción General

## Posicionamiento y Alcance

OptiService se posiciona como una plataforma innovadora que aborda la necesidad creciente de conectar a empleadores y trabajadores independientes de manera eficiente y efectiva. Esta plataforma proporciona una solución integral para la publicación, búsqueda, selección y gestión de trabajos o proyectos, facilitando así la colaboración entre ambas partes.

**Alcance de las Tareas:**

Dadas las restricciones temporales de las prácticas, se ha planificado la implementación de los siguientes módulos principales:

* **Interfaz de Usuario Intuitiva y Fácil de Usar:** Diseñar y desarrollar una interfaz de usuario atractiva y amigable, que permita a los usuarios navegar y utilizar la plataforma de manera sencilla y eficiente.
* **Emparejamiento de Usuarios:** Implementar un motor de emparejamiento de usuarios que facilite la conexión entre empleadores y empleados, considerando criterios como habilidades, ubicación y preferencias.
* **Sistema de Calificación y Reseñas:** Desarrollar un sistema robusto de calificación y reseñas que garantice la transparencia y la calidad de las interacciones entre los usuarios de la plataforma.
* **Funcionalidades de Login y Registro de Usuarios:** Implementar un sistema seguro de autenticación y registro de usuarios, que proteja la privacidad y la seguridad de la información personal.

## Objetivos del diseño

El diseño de OptiService tiene como objetivo principal ofrecer una experiencia de usuario óptima, tanto para empleadores como para empleados, a través de una interfaz intuitiva y fácil de usar. Además, busca garantizar la seguridad, escalabilidad y disponibilidad del sistema para satisfacer las demandas de un mercado laboral dinámico y en constante cambio.

## Características Principales

OptiService se basa en una arquitectura moderna y modular que permite una fácil adaptación a los requisitos cambiantes del mercado. Entre sus características principales se incluyen:

* Interfaz de usuario intuitiva y fácil de usar.
* Emparejamiento de usuarios.
* Sistema de calificación y reseñas para garantizar la calidad de las interacciones.

## Restricciones y limitaciones

El diseño y desarrollo de OptiService está sujeto a ciertas restricciones y limitaciones, que incluyen:

**Duración de las Prácticas:**

Anderson Santana y Alex Pazmiño, practicantes en la carrera de software, tienen una duración de prácticas de 240 horas laborables. Esta limitación temporal condiciona el alcance y la complejidad de las tareas que pueden realizar durante este período.

**Complejidad y Escalabilidad:**

Debido a las limitaciones temporales y de recursos, se priorizará la implementación de funcionalidades esenciales y se procurará diseñar el sistema de manera que pueda escalarse y ampliarse en el futuro, según las necesidades del negocio y las expectativas de los usuarios.

**Duración de las Prácticas:**

Anderson Santana y Alex Pazmiño, practicantes en la carrera de software, tienen una duración de prácticas de 240 horas laborables. Esta limitación temporal condiciona el alcance y la complejidad de las tareas que pueden realizar durante este período.

# Requisitos Atendidos

La motivación y el fundamento de todo lo hecho en el proyecto, no son otros sino los requisitos y necesidades, tanto del cliente como de los futuros usuarios del sistema.

Es por esto, que en esta sección se indican los requisitos atendidos por el diseño o arquitectura que se describirá en las próximas secciones.

## Requisitos Funcionales

|  |  |
| --- | --- |
| **RQF 1** | *Registro de Usuario* |
| **Requisitos asociados** | *RQF 1* |
| **Descripción** | *El sistema permitirá a los usuarios registrarse en el sistema mediante un correo electrónico, se recopila información importante del usuario como nombres, dirección, correo electrónico, número de teléfono, etc.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Se debe considerar otro tipo de información a recopilar en el registro de usuario, como el rol que tendrá dentro del sistema, que un usuario puede tener varios roles, etc.* |

|  |  |
| --- | --- |
| **RQF 2** | *Ingreso de Usuario* |
| **Requisitos asociados** | *RQF 1* |
| **Descripción** | *El sistema permitirá a los usuarios ingresar al sistema mediante su correo electrónico y contraseña previamente registrados.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Se debe considerar validar las credenciales de acceso* |

|  |  |
| --- | --- |
| **RQF 3** | *Registro de publicaciones* |
| **Requisitos asociados** | *RQF 1, RQF 2* |
| **Descripción** | *El sistema permitirá a los usuarios con rol de empleador registrar publicaciones/pedidos. Se debe recopilar información sobre el detalle de la publicación, categoría, ubicación, pago tentativo, número de horas estimadas, etc.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Se debe considerar otro tipo de información que facilite a los usuarios dar a entender lo que necesitan.* |

|  |  |
| --- | --- |
| **RQF 4** | *Modificar publicaciones* |
| **Requisitos asociados** | *RQF 1, RQF 2, RQF 3* |
| **Descripción** | *El sistema permitirá a los usuarios con rol de empleador modificar publicaciones/pedidos previamente registrados en el sistema.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Se debe considerar que información se debe modificar y cual no con respecto a las publicaciones* |

|  |  |
| --- | --- |
| **RQF 5** | *Eliminar publicaciones* |
| **Requisitos asociados** | *RQF 1, RQF 2, RQF 3* |
| **Descripción** | *El sistema permitirá a los usuarios con rol de empleador eliminar publicaciones/pedidos previamente registrados en el sistema.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Se debe considerar en que situaciones el usuario con rol de empleador eliminará sus publicaciones/pedidos.* |

|  |  |
| --- | --- |
| **RQF 6** | *Mostrar publicaciones* |
| **Requisitos asociados** | *RQF 1, RQF 2, RQF 3* |
| **Descripción** | *El sistema permitirá a los usuarios ver las publicaciones de los empleadores en su pantalla de inicio* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Se debe considerar parámetros de búsqueda como la categoría del pedido, ubicación, etc.* |

|  |  |
| --- | --- |
| **RQF 7** | *Postular a publicaciones* |
| **Requisitos asociados** | *RQF 1, RQF 2, RQF 3, RQF 6* |
| **Descripción** | *El sistema permitirá a los usuarios con rol de empleado postular a publicaciones/pedidos previamente registrados en el sistema.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Se debe considerar parámetros de postulación.* |

|  |  |
| --- | --- |
| **RQF 8** | *Seleccionar postulantes* |
| **Requisitos asociados** | *RQF 1, RQF 2, RQF 3, RQF 6, RQF 7* |
| **Descripción** | *El sistema permitirá a los usuarios con rol de empleador seleccionar a los postulantes a sus publicaciones/pedidos previamente registrados en el sistema.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Se deberá mostrar información relevante de los postulantes para que el usuario determine su decisión.* |

|  |  |
| --- | --- |
| **RQF 9** | *Calificación y Retroalimentación* |
| **Requisitos asociados** | *RQF 1, RQF 2, RQF 3, RQF 6, RQF 7, RQF 8, RQF 9* |
| **Descripción** | *El sistema permitirá a los usuarios calificar y emitir comentarios para feedback sobre el contacto realizado por la publicación/pedido.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Para que los usuarios cierren el acuerdo deberán calificar, empleador a empleado y viceversa, para ello será necesario registrar alguna evidencia de que el pedido fue realizado para proceder a calificar y emitir comentarios en caso de que el usuario lo crea conveniente.* |

|  |  |
| --- | --- |
| **RQF 10** | *Modificar perfil* |
| **Requisitos asociados** | *RQF 1, RQF 2* |
| **Descripción** | *El sistema permitirá a los usuarios modificar su perfil.* |
| **Prioridad** | *Media* |
| **Comentarios** | *Se debe considerar información adicional a recopilar, sin tomar en cuenta la información solicitada en el registro.* |

## Requisitos No Funcionales

|  |  |
| --- | --- |
| **RQNF 1** | *Rendimiento* |
| **Requisitos asociados** | *RQNF 1* |
| **Descripción** | *La aplicación debe ser lo suficientemente rápida para manejar búsquedas y transacciones de manera eficiente.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Debe ser escalable para manejar un crecimiento futuro en la cantidad de usuarios y trabajos.* |

|  |  |
| --- | --- |
| **RQNF 2** | *Usabilidad* |
| **Requisitos asociados** | *RQNF 2* |
| **Descripción** | *La interfaz de usuario debe ser intuitiva y fácil de usar tanto para empleadores como para empleados.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *Deben proporcionarse funciones de búsqueda avanzada y filtros para facilitar la búsqueda de trabajos.* |

|  |  |
| --- | --- |
| **RQNF 3** | *Compatibilidad* |
| **Requisitos asociados** | *RQNF 4* |
| **Descripción** | *La aplicación debe ser compatible con una variedad de navegadores web.* |
| **Prioridad** | *Alta* |
| **Comentarios** | *La aplicación debe ser responsive, deberá mostrarse de forma adecuada en los diferentes tamaños de ventanas y dispositivos, ya sea en ordenador o en teléfonos móviles.* |

# Vistas, Capas y Planos

Los sistemas intensivos en software se encuentran formados por un conjunto de *componentes*, que no son más que los elementos listos para ser ejecutados producidos por el proyecto.

Dichos *componentes* se distribuyen sobre los distintos equipos según lo que se detalla en la *vista de despliegue*.

Por otro lado, la *capa lógica* presenta al sistema como un todo, indicando en términos propios de la tecnología utilizada, las partes que lo forman y las relaciones principales entre ellas.

Finalmente se presenta el llamado *modelo de datos*, que contiene la estructura de almacenamiento de información requerida por el sistema aquí descrito.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Capa* | Elementos | *Contiene* | *Se relaciona con* | *Comentarios* |
| *Capa de Presentación* | *Componentes* | *Elementos visuales reutilizables como botones, formularios, barras de navegación, etc* | *Modelos, Servicios API* | *Los componentes interactúan con los modelos y llaman a servicios para obtener datos o realizar acciones.* |
| *Templates* | *Archivos HTML que definen la estructura visual de cada componente.* | *Componentes* | *Los templates utilizan los componentes para definir la apariencia de la interfaz de usuario.* |
| *Estilos* | *Archivos CSS o SCSS que contienen estilos para personalizar la apariencia de los componentes.* | *Componentes* | *Los estilos definen la apariencia visual de los componentes.* |
| *Servicios* | *Servicios API* | *Servicios encargados de realizar peticiones HTTP al backend y procesar las respuestas.* | *Controladores, Modelos de Datos* | *Los servicios API interactúan con el backend para obtener o enviar datos a través de HTTP.* |
| *Servicios de Datos* | *Encargados de interactuar con la base de datos, realizando operaciones CRUD y consultas.* | *Modelos de Entidades, Modelos de Transferencia de Datos* | *Los servicios de datos realizan operaciones CRUD y consultas en la base de datos.* |
| *Servicios de Autenticación y Autorización* | *Gestión de la autenticación y autorización de usuarios.* | *Modelos de Usuarios, Servicios de Datos* | *Los servicios de autenticación y autorización gestionan la identidad y los permisos de los usuarios.* |
| *Servicios de Lógica* | *Servicios que encapsulan lógica de negocio específica, como cálculos o validaciones en el cliente.* | *Componentes, Modelos de Datos* | *Los servicios de lógica proporcionan funciones reutilizables para manejar la lógica de la aplicación.* |
| *Modelos* | *Modelos de Datos* | *Interfaces que definen la estructura de los objetos de datos que se utilizan en la aplicación.* | *Servicios de Datos* | *Los modelos de datos representan la estructura de los objetos de datos manipulados por la aplicación.* |
| *Modelos de Vista* | *Interfaces que representan la estructura de los datos que se muestran en la interfaz de usuario.* | *Componentes Servicios* | *Los modelos de vista definen la estructura de los datos que se muestran en la interfaz de usuario.* |
| *Modelos de Entidades* | *Representan las estructuras de datos de la base de datos y las relaciones entre ellas.* | *Servicios de Datos* | *Los modelos de entidades representan las tablas de la base de datos y sus relaciones.* |
| *Modelos de transferencia de datos* | *Definen cómo se transmiten los datos entre el frontend y el backend, generalmente en formato JSON.* | *Servicios API, Modelos de Datos* | *Los modelos de transferencia de datos definen la estructura de los objetos de datos transmitidos entre el frontend y el backend.* |
| *Controladores* | *Controladores de Rutas* | *Define cómo responder a las solicitudes HTTP para diferentes endpoints de la API.* | *Servicios API, Modelos de Datos* | *Servicios API, Modelos de Datos* |
| *Controladores de lógica* | *Llevan a cabo la lógica específica para cada solicitud HTTP, como la validación de datos o la llamada a servicios.* | *Servicios de Lógica, Servicios de Datos* | *Los controladores de lógica llevan a cabo la lógica específica para cada solicitud HTTP recibida.* |

## Resumen de Capas

## Vista Lógica

La vista lógica en el contexto de una aplicación web se refiere a la capa que se encarga de la lógica de presentación y la interacción con los datos. En una arquitectura MVC (Modelo-Vista-Controlador), la vista lógica corresponde a la parte del controlador que se encarga de procesar las solicitudes del usuario, interactuar con los modelos de datos y generar la respuesta que se enviará de vuelta al cliente.

**Controladores (Controllers):**

En los commits, se pueden identificar archivos que están relacionados con la lógica de los controladores. Estos controladores manejan las solicitudes HTTP y generan respuestas en función de esas solicitudes. Por ejemplo, los nombres de archivo como controller.js o handlers.js pueden indicar la presencia de controladores.

Los controladores se encargan de interactuar con los modelos de datos y realizar operaciones como crear, leer, actualizar y eliminar (CRUD) en la base de datos.

En los commits, es posible encontrar cambios relacionados con la implementación de controladores para diferentes rutas de la API, como la creación de nuevas rutas (routes.js) o la actualización de rutas existentes para manejar nuevas funcionalidades.

**Rutas (Routes):**

Las rutas definen cómo se mapean las solicitudes HTTP a los controladores correspondientes. En los commits, es posible identificar archivos que definen las rutas de la API, como routes.js o api.js.

Cada ruta está asociada a un controlador específico que maneja las solicitudes HTTP que llegan a esa ruta. Los cambios en estos archivos pueden indicar la adición de nuevas rutas o la modificación de rutas existentes para incluir nuevas funcionalidades.

**Interacción con la Base de Datos:**

Los controladores interactúan con los modelos de datos para realizar operaciones en la base de datos. En los commits, es posible identificar cambios relacionados con la implementación de operaciones CRUD en los modelos de datos.

Los archivos relacionados con los modelos de datos, como models.js o schemas.js, pueden indicar la estructura y el esquema de los datos almacenados en la base de datos.

**Respuestas HTTP:**

Los controladores generan respuestas HTTP en función de las solicitudes recibidas. Estas respuestas pueden estar en formato JSON u otro formato de datos adecuado. En los commits, es posible identificar cambios relacionados con la generación de respuestas HTTP en los controladores

LA vista lógica se organiza en controladores que manejan las solicitudes HTTP, rutas que definen cómo se accede a estos controladores y modelos de datos que interactúan con la base de datos:

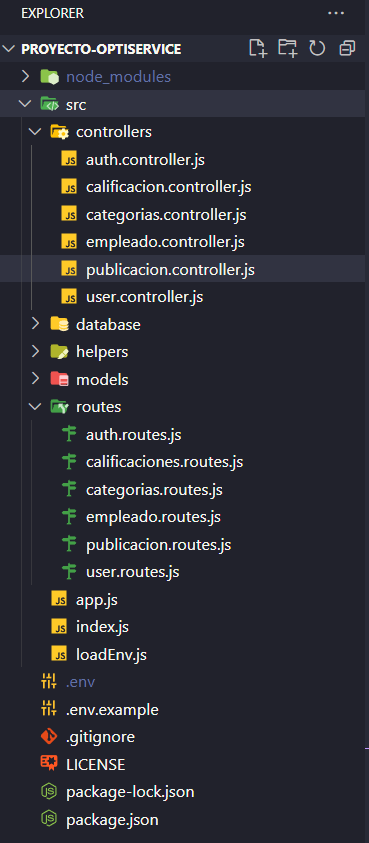


Ilustración 1: Estructura del Backend

En la Ilustración 1 se da a conocer la estructura que posee el Backend, el cual se compone de Controladores, Rutas y Models.

## Vista de Componentes

La vista de componentes en una aplicación Angular se organiza en componentes individuales que encapsulan partes específicas de la interfaz de usuario y su funcionalidad asociada. Estos componentes pueden agruparse y organizarse en módulos, y pueden hacer uso de servicios y directivas para proporcionar funcionalidades compartidas y comportamientos específicos en la aplicación.

**Componentes (Components):**

En Angular, los componentes son los bloques de construcción fundamentales de la interfaz de usuario. Cada componente encapsula una parte específica de la interfaz de usuario y su funcionalidad asociada.

Los commits pueden incluir la creación de nuevos componentes, identificados por nombres de archivos que terminan en .component.ts, como login.component.ts, content.component.ts, etc.

Los cambios en los archivos de los componentes pueden indicar actualizaciones en la lógica de presentación y comportamiento de diferentes partes de la aplicación, como la pantalla de inicio, la barra de navegación, formularios de registro, etc.

Además, es posible encontrar cambios en archivos HTML y CSS relacionados con los componentes, lo que indica actualizaciones en la estructura visual y el estilo de los componentes.

**Módulos (Modules):**

Los módulos en Angular agrupan componentes relacionados y otros artefactos, como directivas y servicios, en unidades funcionales.

Los commits pueden incluir la creación o modificación de módulos, identificados por nombres de archivos que terminan en .module.ts, como app.module.ts, auth.module.ts, etc.

Los módulos definen las dependencias y las configuraciones de la aplicación, y proporcionan un contexto para la agrupación y la organización de los componentes.

**Servicios (Services):**

Los servicios en Angular son clases que proporcionan funcionalidades compartidas en toda la aplicación.

Se incluyen la creación y modificación de servicios, identificados por nombres de archivos que terminan en .service.ts, como auth.service.ts, api.service.ts, etc.

Los servicios encapsulan la lógica de negocio y la lógica de acceso a datos que pueden ser utilizadas por varios componentes.

**Directivas (Directives):**

Las directivas en Angular son marcadores en el DOM que le dicen al framework Angular cómo comportarse con respecto a un elemento DOM existente.

Incluyen la creación de directivas, identificados por nombres de archivos que terminan en .directive.ts, como custom.directive.ts, etc.

Las directivas pueden utilizarse para manipular el DOM, agregar comportamientos específicos a los elementos DOM y reutilizar la funcionalidad en varios componentes.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración 2: Estructura del Frontend

En la Ilustración 2 Se da a conocer la Estructura que tiene el Frontend el cual se estructura por la carpeta modules, dentro de los cuales se encuentran los diferentes componentes como lo son el Login, El registro, la pagina principal, entre otros, También se posee la carpeta de servicios que contiene los diferentes servicios que se encargan de hacer las peticiones la Backend y por ultimo se tiene las rutas, que serán las encargadas de redirigir hacia los componentes cuando se requiera.

## Vista de Datos

**Base de Datos PostgreSQL:**

La aplicación utiliza PostgreSQL como sistema de gestión de bases de datos relacional.

PostgreSQL ofrece un sólido soporte para la gestión de datos estructurados, lo que te permite almacenar y manipular la información necesaria para tu aplicación.

Puedes aprovechar las características avanzadas de PostgreSQL, como los tipos de datos personalizados y las funciones almacenadas, para diseñar una base de datos que se adapte a las necesidades de la aplicación.

**Backend (API REST):**

La aplicación backend actúa como una interfaz entre la base de datos y el frontend, proporcionando endpoints API RESTFULL para que el frontend pueda interactuar con los datos.

Los commits en el repositorio de backend muestran implementaciones relacionadas con la lógica de acceso a datos, como la creación de modelos, controladores y rutas API.

Las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) están presentes en la lógica del backend para permitir la manipulación de datos en la base de datos.

**Frontend (Angular):**

El frontend de la aplicación, desarrollado en Angular, consume los endpoints API REST proporcionados por el backend para obtener y manipular datos.

Los commits en el repositorio de frontend muestran implementaciones relacionadas con la visualización y manipulación de datos en la interfaz de usuario, como la creación de componentes, servicios y rutas.

Se utilizan servicios HTTP para realizar solicitudes al backend y recibir respuestas de datos, que luego se presentan en la interfaz de usuario utilizando plantillas HTML y estilos CSS.

**Interacción entre Backend y Frontend:**

El backend y el frontend trabajan juntos para proporcionar una experiencia de usuario fluida y coherente.

El backend se encarga de la lógica de negocio y el acceso a la base de datos, mientras que el frontend se encarga de presentar los datos de manera atractiva y fácilmente accesible para los usuarios.

# Detalles de Implementación

**Backend (API REST):**

Implementación de modelos: Define los modelos de datos que representan las entidades en tu base de datos. Estos modelos están vinculados a tablas específicas en la base de datos y definen la estructura de los datos que se pueden almacenar y recuperar.

Controladores: Los controladores son responsables de manejar las solicitudes HTTP entrantes, procesar la lógica de negocio correspondiente y devolver las respuestas adecuadas. En tu caso, los controladores pueden realizar operaciones como crear, leer, actualizar y eliminar datos en la base de datos.

Rutas API: Define las rutas de acceso a los diferentes endpoints de tu API REST. Estas rutas mapean las solicitudes HTTP entrantes a los métodos correspondientes en los controladores, permitiendo la interacción con los datos a través de la API.

**Frontend (Angular):**

Servicios: Los servicios en Angular se utilizan para encapsular la lógica relacionada con el acceso a datos y la comunicación con el backend a través de HTTP. En tu caso, puedes tener servicios que manejen las solicitudes HTTP para obtener y enviar datos al backend.

Componentes: Los componentes en Angular representan las diferentes partes de la interfaz de usuario y contienen la lógica para interactuar con los datos y responder a las acciones del usuario. Puedes tener componentes para mostrar listas de datos, formularios para agregar o editar datos, y otros elementos de la interfaz de usuario.

Enrutamiento: El enrutamiento en Angular se utiliza para definir la navegación dentro de la aplicación y asociar rutas de URL específicas con componentes particulares. Esto permite la navegación entre diferentes vistas y la carga dinámica de componentes basados en la URL actual.

**Interacción entre Backend y Frontend:**

Comunicación HTTP: El frontend realiza solicitudes HTTP al backend para obtener y enviar datos utilizando los servicios proporcionados por Angular. Esto se hace típicamente utilizando los métodos HTTP estándar como GET, POST, PUT y DELETE.

Formato de datos: La comunicación entre el backend y el frontend generalmente implica el intercambio de datos en formato JSON (JavaScript Object Notation), que es un formato ligero y fácil de leer. Angular proporciona utilidades integradas para trabajar con datos JSON tanto en las solicitudes HTTP entrantes como salientes.

## Plataformas y Lenguajes

|  |  |  |  |
| --- | --- | --- | --- |
| *Se requiere* | *Versión* | *Debido a* | *Comentarios* |
| *Angular* | *16.2.1* | *Framework basado en TypeScript para el desarrollo Frontend* | *N/A* |
| *Node* | *18.10.0* | *Entorno de ejecución de JavaScript que se utiliza para desarrollar aplicaciones del lado del servidor* | *Se recomienda instalar NVM (Node Version Manager)* |
| *Express.js* | *4.18.2* | *Framework basado en node.js para el desarrollo Backend* | *N/A* |
| *PostgreSQL* | *15* | *Sistema de gestión de bases de datos relacionales* | *N/A* |
| *Visual Studio Code* | *1.86.1* | *Entorno de desarrollo* | *N/A* |

## Organización del proyecto

Obtención e instalación del paquete de fuentes

*Para obtener e instalar el paquete de fuentes para los proyectos de frontend y backend que mencionaste, sigue estos pasos:*

***Para el Frontend:***

*Obtención de fuentes desde el repositorio de GitHub:*

* *Dirígete al repositorio del frontend en GitHub: https://github.com/Alxsnc/Optiserve-Front.git*
* *Haz clic en el botón "Code" y luego selecciona HTTPS, se nos mostrara el link del repositorio*
* *En una carpeta abrimos el cmd y ponemos el comando git.clone [URL del repositorio] .*
* *Arrastramos la carpeta hacia el Visual Studio y ejecutamos el comando npm install para instalar todas las dependencias del proyecto.*
* *Para ejecutar el proyecto, en una nueva terminal ingresamos el comando ng serve -o*

***Para el Backend:***

*Obtención de fuentes desde el repositorio de GitHub:*

* *Accede al repositorio del backend en GitHub: https://github.com/Juvth/Proyecto-OptiService.git*
* *Haz clic en el botón "Code" y luego selecciona "HTTPS" y copia el link del repositorio*
* *Abrimos el cmd en una carpeta y ponemos el comando git clone [URL del repositorio] .*
* *Arrastramos la carpeta al icono de Visual Studio Code para abrir el proyecto*
* *Abrimos una terminal y ejecutamos el comando npm install para instalar las dependencias del proyecto.*
* *Para ejecutar el proyecto, en la terminal colocamos el comando npm run dev*

*Una vez que hayas seguido estos pasos, habrás obtenido las fuentes de ambos proyectos desde GitHub y habrás instalado todas las dependencias necesarias para cada uno. Ahora estás listo para trabajar con los proyectos de frontend y backend en tu ambiente de desarrollo.*

## Referencias

## Insumos

David Esteller Ortega. *“Manual para organizar cooperativas”*. Vadell Hermanos Editores, C.A. Valencia, Venezuela. ISBN: 980-212-099-5

Alberto G. Alexander Servat. *“Manual para documentar sistemas de calidad”*. Prentice Hall, México. 1998. ISBN: 970-17-0185-2

Luís T. Díez de Castro y Joaquín López Pascual. *“Dirección Financiera”*. Prentice Hall, Madrid. 2001. ISBN: 84-205-3066-2

## Glosario de términos

* ***Repositorio de GitHub:*** *Un repositorio de GitHub es un espacio donde se almacena y se gestiona el código fuente de un proyecto. Permite a los desarrolladores colaborar, realizar un seguimiento de cambios, revisar el código y gestionar problemas.*
* ***Frontend:*** *El frontend se refiere a la parte de una aplicación o sistema que interactúa directamente con el usuario. Por lo general, consiste en la interfaz de usuario, que incluye la presentación y la interacción con la aplicación a través de elementos como botones, formularios, y otros componentes visuales.*
* ***Backend:*** *El backend se refiere a la parte de una aplicación o sistema que se encarga del procesamiento y almacenamiento de datos, así como de la lógica de negocio. Maneja las solicitudes del cliente, accede a la base de datos, realiza cálculos y devuelve respuestas al frontend.*
* ***Git****: Git es un sistema de control de versiones distribuido utilizado para rastrear cambios en el código fuente durante el desarrollo de software. Permite a los equipos de desarrollo colaborar de manera efectiva, gestionar ramas de código, fusionar cambios y revertir a versiones anteriores si es necesario.*
* ***Terminal o línea de comandos:*** *Es una interfaz de texto que permite a los usuarios enviar comandos directamente a un sistema operativo. Se utiliza para realizar diversas tareas, como navegar por el sistema de archivos, ejecutar programas y realizar operaciones en un entorno de desarrollo.*
* ***npm:*** *npm es el administrador de paquetes de Node.js, utilizado para instalar, gestionar y compartir paquetes de código JavaScript. Permite a los desarrolladores instalar dependencias, como bibliotecas y frameworks, necesarias para sus proyectos.*
* ***Dependencias:*** *Las dependencias son paquetes de software externos utilizados por un proyecto para proporcionar funcionalidades específicas. Estas dependencias deben ser instaladas para que el proyecto funcione correctamente. npm es comúnmente utilizado para gestionar las dependencias en proyectos de Node.js.*
* ***package.json****: Es un archivo de configuración utilizado en proyectos de Node.js para definir metadatos del proyecto, como el nombre, la versión, las dependencias y los scripts de inicio. Contiene información esencial sobre el proyecto y permite a npm gestionar las dependencias y ejecutar comandos específicos del proyecto.*