INFORME ARQUITECTURA DE SOFTWARE

Presentado a:

Pablo Antonio Ortiz

Integrantes:

Diana Katherine Vásquez Escobar

Royer Daniel Solarte

Ana María Sarria Arce

María Fernanda Anaya

Ricardo Erazo Muñoz

Esteban Daza Muñoz

Joshua Orozco Tobar

N° de Ficha:

2770242

Tecnólogo en Análisis y Desarrollo de Software

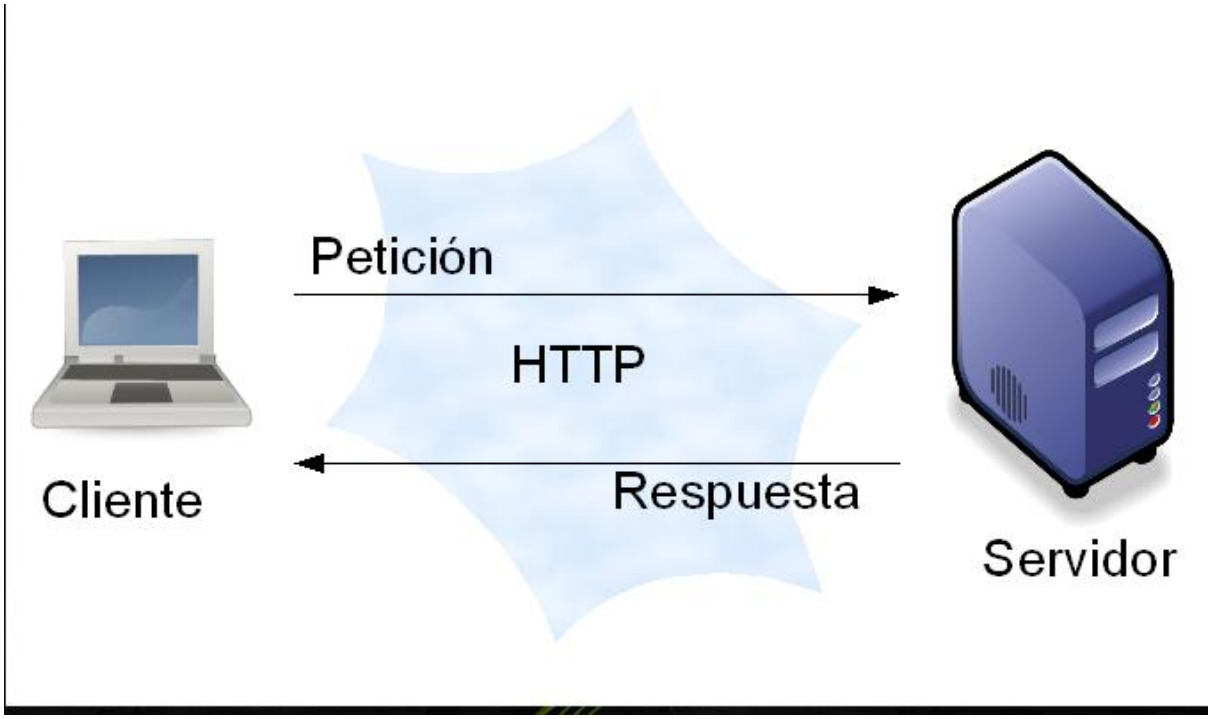
Servicio Nacional de Aprendizaje SENA

Centro de Teleinformática y Producción Industrial

Regional Cauca

Popayán, día 10 de febrero del año 2025

TABLA DE CONTENIDO



El aplicativo web y móvil el cual es un sistema de información para la **Asociación de** Recicladores de Oficio Goleros (AROG) se basa en una arquitectura cliente-servidor debido a su capacidad para establecer una relación clara entre los usuarios finales (clientes) y el servidor central. En este modelo, los clientes realizan solicitudes al servidor, quien se encarga de procesarlas y devolver respuestas. Esta estructura no solo distribuye las responsabilidades entre los componentes del sistema, sino que también garantiza eficiencia, escalabilidad y seguridad,

En una arquitectura cliente-servidor, los roles están definidos. Por un lado, el **cliente** representa los dispositivos o aplicaciones que interactúan directamente con los usuarios, como ciudadanos, recolectores y administradores. En el caso de AROG, el cliente está conformado por una aplicación web y móvil, desarrollada con tecnologías como Next.js y Flutter, que permite a los usuarios enviar solicitudes, visualizar información y realizar diversas operaciones. Por otro lado, el **servidor** actúa como el centro del sistema, procesando las solicitudes, gestionando la lógica de negocio y manejando el acceso a la base de datos. El backend, construido con java script, Node.js y Express, se encarga de validar y responder a las peticiones de manera eficiente.

El flujo de interacción entre el cliente y el servidor es otro aspecto clave de esta arquitectura. Cada acción realizada por un usuario genera una solicitud que es procesada por el servidor. Por ejemplo, cuando un ciudadano consulta las rutas de recolección, la aplicación cliente envía una solicitud al servidor, quien accede a la base de datos, obtiene la información requerida y la devuelve al cliente para su visualización. De manera similar, cuando un recolector registra la recolección de residuos, el cliente envía un formulario con los datos al servidor, quien los valida y almacena en la base de datos. Este proceso también se aplica cuando un administrador asigna una nueva tarea, ya que la solicitud es enviada al servidor, quien actualiza la base de datos y envía notificaciones a los recolectores.

La comunicación entre el cliente y el servidor se realiza a través de una **API REST**, que actúa como intermediaria para asegurar que las solicitudes se procesen correctamente. Esta API recibe las solicitudes del cliente, interpreta y gestiona la lógica de negocio, consulta o modifica la base de datos, y retorna respuestas en formato JSON para que el cliente las interprete. Este modelo no solo facilita la interacción entre los componentes, sino que también asegura un flujo de datos eficiente y seguro.

Entre los principales beneficios de la arquitectura cliente-servidor en el sistema de información para AROG se encuentran la escalabilidad, ya que se pueden agregar más clientes sin afectar el rendimiento del sistema; la seguridad, pues el servidor central controla el acceso a los datos mediante autenticación y encriptación; la eficiencia, dado que el cliente se enfoca en la interfaz y experiencia de usuario mientras el servidor maneja el procesamiento de datos; y la mantenibilidad, ya que se pueden actualizar la aplicación web/móvil o el backend sin afectar la otra parte.

PATRON DE ARQUITECTURA

```
project-root/
├── public/                 # Archivos estáticos (favicon, imágenes)
├── src/                    # Código fuente
│   ├── models/             # Modelos (datos)
│   ├── views/              # Vistas (UI)
│   │   ├── pages/          # Páginas completas
│   │   ├── components/     # Componentes reutilizables
│   │   └── layouts/        # Plantillas de diseño
│   ├── controllers/        # Controladores (lógica de negocio)
│   ├── styles/             # Estilos (Tailwind y globales)
│   ├── utils/              # Funciones auxiliares
│   └── config/             # Configuraciones del proyecto
├── .env                    # Variables de entorno
├── package.json            # Dependencias del proyecto
├── README.md               # Documentación
└── tailwind.config.js      # Configuración de Tailwind CSS
```

El patrón de arquitectura seleccionado para el desarrollo del Sistema de Información AROG es el Modelo Vista Controlador (MVC), una elección que se fundamenta en su capacidad para organizar el código en capas separadas, lo que facilita la mantenibilidad, escalabilidad y reutilización del sistema. Este patrón es especialmente adecuado para un sistema que debe gestionar la interacción de diferentes tipos de usuarios, como ciudadanos, recolectores y administradores, quienes realizan solicitudes al servidor para consultar rutas, registrar actividades y administrar tareas entre otras actividades.

Implementamos es patrón porque organiza la lógica en módulos independientes, evitando código desordenado y facilitando el mantenimiento. Además, permite una interacción fluida entre el cliente y el servidor mediante una API REST, gestionando las solicitudes de manera eficiente. Este patrón también favorece la escalabilidad y la incorporación de nuevas funcionalidades sin afectar la estructura existente. Al separar las responsabilidades en Modelo, Vista y Controlador, MVC se adapta perfectamente a la arquitectura cliente-servidor del proyecto.

Este se estructura en tres componentes principales: el Modelo, que representa la base de datos y normativas de operación, utilizando MySQL junto con Prisma ORM para gestionar datos como información de usuarios, rutas y registros de recolección; la Vista, desarrollada en Next.js para la versión web y Flutter para la móvil, que proporciona la interfaz de usuario para ciudadanos, recolectores y administradores; y el Controlador, implementado con Node.js y Express, que actúa como intermediario entre la vista y el modelo, manejando solicitudes, validando permisos, procesando lógica de negocio y devolviendo respuestas en formato JSON. Esta separación de responsabilidades garantiza un sistema organizado, escalable y fácil de mantener.

DISEÑO DE ARQUITECTURA

El sistema de información de la Asociación de Recicladores de Oficio Goleros (AROG) se diseñará con una arquitectura cliente-servidor basada en el patrón MVC (Modelo-Vista-Controlador). Esto garantizará una separación clara de responsabilidades, facilitando la escalabilidad, mantenibilidad y seguridad del sistema.

El sistema estará compuesto por una aplicación web y móvil, una API RESTful y una base de datos en la nube. El diseño de arquitectura se enfocará en definir cómo se implementará cada componente, qué tecnologías se utilizarán y cómo interactuarán entre sí.

El diseño de arquitectura se basa en un modelo en capas con cuatro componentes principales: Capa de Presentación, Capa de Lógica de Negocio, Capa de Datos y Infraestructura en la Nube. La Capa de Presentación estará a cargo de la interfaz con los usuarios, utilizando tecnologías como Next.js y Flutter. La Capa de Lógica de Negocio procesará las solicitudes de los usuarios y manejará la lógica del sistema, utilizando Node.js con Prisma como ORM para la interacción con la base de datos y JWT para autenticación segura.

La elección de Prisma se debe a su capacidad para proporcionar una capa de abstracción entre la aplicación y la base de datos, permitiendo una mayor flexibilidad y escalabilidad. Además, Prisma ofrece una sintaxis más simple y elegante para la definición de modelos y consultas, lo que facilita el desarrollo y mantenimiento de la aplicación.

La Capa de Datos almacenará la información del sistema, utilizando MySQL como base de datos relacional. Finalmente, la Infraestructura en la Nube asegurará que el sistema sea accesible en todo momento, utilizando tecnologías como EC2, RDS y S3 de AWS.

CONCLUSION

El diseño de arquitectura del aplicativo web y movi de AROG combina una arquitectura cliente-servidor con el patrón MVC, asegurando una separación clara de responsabilidades y facilitando la escalabilidad y mantenibilidad del software. La implementación con Next.js y Flutter para el frontend, Node.js con Prisma para el backend y MySQL en AWS como base de datos, garantiza un sistema eficiente, seguro y accesible desde cualquier dispositivo. Esta estructura modular permite gestionar rutas, recolectores y registros de residuos de manera óptima, mejorando la productividad de la Asociación AROG y promoviendo una gestión sostenible de los residuos en Popayán.

GLOSARIO

- API (Interfaz de Programación de Aplicaciones): Conjunto de reglas y herramientas que permiten la comunicación entre diferentes sistemas de software.

- Arquitectura Cliente-Servidor: Modelo en el que los clientes (aplicaciones web y móviles) solicitan datos o servicios a un servidor central.

- AWS (Amazon Web Services): Plataforma de servicios en la nube utilizada para alojar el backend, la base de datos y el almacenamiento de archivos del sistema.

- Backend: Parte del sistema encargada de procesar la lógica de negocio y manejar las solicitudes del cliente a través de una API.

- Base de Datos Relacional (MySQL): Sistema de almacenamiento estructurado que organiza la información en tablas relacionadas.

- Controlador (Controller): Componente del patrón MVC que maneja la lógica de negocio y gestiona la comunicación entre la vista y el modelo.

- Frontend: Interfaz gráfica con la que interactúan los usuarios, implementada en React.js para la versión web y Flutter para la versión móvil.

- JWT (JSON Web Token): Mecanismo de autenticación basado en tokens que permite la validación segura de usuarios en el sistema.

- Microservicio: Pequeña aplicación independiente que cumple una función específica dentro de un sistema distribuido, usado en arquitecturas escalables.

- MVC (Modelo-Vista-Controlador): Patrón de diseño que separa la lógica del negocio (Modelo), la presentación de la información (Vista) y la gestión de las interacciones del usuario (Controlador).

- Node.js: Entorno de ejecución de JavaScript utilizado en el backend para manejar solicitudes de los clientes de manera eficiente.

- ORM (Object-Relational Mapping): Técnica que permite interactuar con bases de datos mediante objetos en código, utilizada con Prisma en este proyecto.

- Patrón de Diseño: Solución estructurada y reusable para problemas comunes en el desarrollo de software, como el patrón MVC utilizado en este sistema.

- RESTful API: Arquitectura de comunicación entre sistemas que permite a las aplicaciones intercambiar datos mediante solicitudes HTTP.

- Rutas de Recolección: Zonas y horarios en los que los recolectores realizarán la recolección de residuos orgánicos, gestionadas desde la aplicación.

- Scalabilidad: Capacidad del sistema para crecer y manejar un mayor número de usuarios y datos sin afectar su rendimiento.

- Prisma: ORM para Node.js que facilita la gestión de bases de datos MySQL mediante código JavaScript en lugar de consultas SQL directas.

- Token: Clave generada automáticamente que se usa para validar y autorizar acciones de los usuarios en el sistema.

- Vista (View): Componente del patrón MVC que representa la interfaz gráfica del usuario y muestra la información procesada por el sistema.

WEBGRAFIA

1.Express.js - Framework para Node.js

https://expressjs.com/

2.Prisma - ORM para Node.js

https://www.prisma.io/docs

3.React.js - Biblioteca para Frontend

https://react.dev/

4.Flutter - Framework para Aplicaciones Móviles

https://docs.flutter.dev/

5.JWT (JSON Web Token) - Seguridad y Autenticación

https://jwt.io/introduction

6.RESTful API - Diseño y Principios

https://restfulapi.net/

7.AWS (Amazon Web Services) - Servicios en la Nube

https://aws.amazon.com/

8.MVC (Modelo-Vista-Controlador) - Patrón de Diseño

https://developer.mozilla.org/es/docs/Glossary/MVC

8.MySQL - Base de Datos Relacional

https://dev.mysql.com/doc/