**Documento de Arquitectura de Software**

***Sistema Integrado de Gestión para el Laboratorio De Análisis Agronómicos del Instituto Tecnológico de Costa Rica***

***Fecha: 20/10/2023***

**Identificación del Proyecto: *SIGLAAITCR-01***

**Versión: *1.0***

Contenido

[Control de Versiones 3](#_Toc148721432)

[Datos del Proyecto 3](#_Toc148721433)

[Nombre del Proyecto Siglas del Proyecto 4](#_Toc148721434)

[Arquitectura de Software 4](#_Toc148721435)

[Introducción de la Arquitectura 5](#_Toc148721436)

[Objetivos de la Arquitectura 5](#_Toc148721437)

[Visión General de la Arquitectura 6](#_Toc148721438)

[Patrones de Diseño 7](#_Toc148721439)

[Decisiones de Diseño 8](#_Toc148721440)

[Vista de Despliegue 10](#_Toc148721441)

[Conclusiones 11](#_Toc148721442)

# Control de Versiones

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Control de Versiones | | | | | |
| *Versión* | *Hecha por* | *Revisada por* | *Aprobada por* | *Fecha* | *Motivo* |
| 1.0 | Fabricio Alexander Porras Morera. | Rachell Jarquín Quesada.  Carlos Eduardo Solís Mora. | Rachell Jarquín Quesada.  Fabricio Alexander Porras Morera.  Carlos Eduardo Solís Mora. | 20/10/2023 | Versión inicial |

# Datos del Proyecto

|  |  |
| --- | --- |
| Nombre del Proyecto Siglas del Proyecto | |
| Sistema Integrado de Gestión para el Laboratorio De Análisis Agronómicos del Instituto Tecnológico de Costa Rica | SIGLAAITCR |

# Arquitectura de Software

## Introducción de la Arquitectura

La arquitectura de software desempeña un papel fundamental en el éxito del proyecto enfocado en la optimización de los procesos operativos y la eficiencia en la gestión de datos en el Laboratorio de Análisis Agronómicos. Este proyecto tiene como objetivo abordar los desafíos inherentes al manejo de información y mejorar la calidad de los servicios ofrecidos por el laboratorio a través de la creación e implementación de un Sistema Integrado de Gestión.

La arquitectura de software es el marco estructural que define la organización de los componentes del sistema, sus relaciones y sus propiedades. La arquitectura de software proporciona la base sobre la cual se construirá el Sistema Integrado de Gestión del Laboratorio, que permitirá digitalizar y automatizar diversas etapas del proceso, desde la captura de datos hasta el procesamiento y presentación de resultados.

La importancia de la arquitectura de software radica en su capacidad para influir en la mejora global de la eficiencia y la calidad de los servicios del laboratorio. La arquitectura define cómo los diferentes módulos, tecnologías y componentes se integran para lograr los objetivos del proyecto. Además, proporciona una estructura sólida que simplifica la gestión interna y la interacción con los clientes.

A continuación, se explorará en detalle la arquitectura de software utilizada para este proyecto, que se basa en el framework React en el front-end y express con node.js en el back-end. También se emplea un ORM llamado Prisma para facilitar la interacción con la base de datos PostgreSQL. Esta elección de tecnologías y la manera en que se integran son cruciales para el éxito del proyecto y serán examinadas en profundidad en las secciones posteriores. La arquitectura de software no solo define cómo se construirá el sistema, sino también cómo se escalará, mantendrá y evolucionará para satisfacer las necesidades cambiantes del laboratorio y sus clientes.

## Objetivos de la Arquitectura

La arquitectura debe ser escalable para adaptarse al crecimiento futuro de los Laboratorios de Análisis Agronómicos. Debe permitir la incorporación de nuevos módulos y funcionalidades sin interrupciones en el servicio y con un rendimiento óptimo.

El sistema debe ser altamente eficiente en términos de rendimiento. Debe ser capaz de manejar grandes volúmenes de datos y transacciones, garantizando tiempos de respuesta rápidos y una experiencia de usuario fluida.

La seguridad de los datos y la privacidad de los usuarios son de suma importancia. La arquitectura debe incluir medidas de seguridad robustas para proteger los datos sensibles y prevenir amenazas de seguridad, como acceso no autorizado o ataques.

La arquitectura debe ser flexible y modular, lo que permite la fácil adición o modificación de componentes del sistema. Esto facilita la adaptación a las cambiantes necesidades del laboratorio y la incorporación de nuevas tecnologías.

La arquitectura debe ser fácil de mantener y actualizar. Se deben seguir las mejores prácticas de desarrollo y documentación para garantizar que futuras modificaciones o mejoras sean manejables sin interrupciones significativas en el servicio.

La arquitectura debe priorizar la experiencia del usuario. Debe ser intuitiva, receptiva y fácil de usar para garantizar que tanto el personal del laboratorio como los clientes tengan una experiencia satisfactoria.

Se deben implementar estrategias sólidas de respaldo y recuperación de datos para garantizar la disponibilidad continua de la información crítica en caso de fallos o desastres.

## Visión General de la Arquitectura

La arquitectura de software se basa en una estructura moderna y flexible que se compone de dos componentes principales: el front-end y el back-end. Estos componentes trabajan en conjunto para proporcionar una solución integral que digitaliza y automatiza las operaciones del laboratorio.

* **Front-end (Interfaz de Usuario):** El front-end es la parte visible del sistema y se encarga de la interacción con los usuarios, tanto internos como externos. Está construido utilizando el framework React, lo que garantiza una experiencia de usuario potente y altamente interactiva. Los componentes principales del front-end incluyen:

**Interfaces de Usuario (UI):** Aquí se encuentran las diversas pantallas, formularios y elementos visuales que los usuarios utilizan para interactuar con el sistema. Estas interfaces están diseñadas para ser intuitivas y amigables.

**Gestión de Datos en el Cliente:** El front-end se encarga de la captura y validación de datos en la interfaz del usuario antes de enviarlos al back-end para su procesamiento. También puede incluir funciones de visualización de resultados y generación de informes.

**Comunicación con el Back-end:** El front-end se comunica con el back-end a través de solicitudes HTTP para enviar y recibir datos. Esto permite una experiencia de usuario en tiempo real y la actualización dinámica de información.

* **Back-end (Lógica de Negocio):** El back-end es la parte del sistema que maneja la lógica de negocio, el procesamiento de datos y la gestión de bases de datos. Está construido utilizando Express con Node.js y se integra con una base de datos PostgreSQL utilizando el ORM Prisma. Los componentes principales del back-end incluyen:

**Rutas y Controladores:** Define las rutas de API que el front-end utiliza para interactuar con el sistema. Los controladores gestionan las solicitudes y las respuestas, realizando operaciones como la validación de datos, el acceso a la base de datos y la lógica de negocio.

**Base de Datos (PostgreSQL):** El back-end se conecta a una base de datos PostgreSQL para almacenar y recuperar datos. Se utiliza el ORM Prisma para simplificar la interacción con la base de datos, lo que facilita la gestión de datos complejos.

## Patrones de Diseño

Los sistemas se benefician de la aplicación de varios patrones de diseño que ayudan a estructurar y optimizar el sistema. A continuación, se identifican y explican algunos de los patrones de diseño utilizados:

**Patrón Modelo-Vista-Controlador (MVC):** Este patrón se aplica principalmente en el front-end y el back-end. En el front-end, React proporciona una estructura MVC, donde el modelo representa los datos, la vista muestra la interfaz de usuario y el controlador maneja las interacciones del usuario. En el back-end, Express y los controladores definen la lógica de negocio y las rutas, siguiendo una separación clara de responsabilidades.

**Patrón de Capas:** La arquitectura sigue un enfoque de capas que separa claramente las responsabilidades. Por ejemplo, el front-end se centra en la presentación y la interacción con el usuario, mientras que el back-end se encarga de la lógica de negocio y la gestión de datos. Esta separación facilita la escalabilidad y el mantenimiento.

**Patrón de API RESTful:** La comunicación entre el front-end y el back-end se basa en el patrón de arquitectura REST (Representational State Transfer). Las solicitudes HTTP siguen convenciones REST, lo que permite una comunicación eficiente y predecible entre los componentes del sistema.

**Patrón ORM (Mapeo Objeto-Relacional):** Prisma se utiliza como un ORM que proporciona una capa de abstracción entre la aplicación y la base de datos PostgreSQL. Este patrón simplifica la interacción con la base de datos al mapear objetos de la aplicación a tablas de la base de datos, lo que reduce la complejidad en la gestión de datos.

**Patrón de Singleton:** En algunos casos, se pueden aplicar patrones Singleton para garantizar que ciertos objetos, como conexiones de bases de datos, sean únicos en toda la aplicación. Esto puede ayudar a optimizar el uso de recursos y garantizar la coherencia de datos.

## Decisiones de Diseño

La arquitectura de software para la optimización de procesos operativos y la gestión eficiente de datos en los Laboratorios de Análisis Agronómicos se basa en una serie de decisiones de diseño clave que influyen en la estructura y el funcionamiento del sistema. Estas decisiones se tomaron considerando los objetivos del proyecto y las necesidades específicas del laboratorio. A continuación, se explican algunas de las decisiones de diseño más importantes:

**Elección de Tecnologías Front-end y Back-end:** La elección de React en el front-end y Express con Node.js en el back-end se basó en la compatibilidad entre estas tecnologías. Node.js permite la ejecución de JavaScript tanto en el front-end como en el back-end, lo que facilita la gestión de datos y la construcción de interfaces de usuario interactivas.

**Uso de Prisma como ORM:** La elección de Prisma como ORM simplifica la interacción con la base de datos PostgreSQL. Esto reduce la complejidad de la gestión de datos y acelera el desarrollo al proporcionar una capa de abstracción que mapea objetos de la aplicación a tablas de la base de datos.

**Arquitectura de Capas:** La arquitectura de capas se implementó para separar claramente las responsabilidades entre el front-end y el back-end. Esta decisión facilita la escalabilidad y el mantenimiento del sistema, ya que cada capa se enfoca en tareas específicas.

**Comunicación a través de API RESTful:** Se optó por utilizar una API RESTful para la comunicación entre el front-end y el back-end. Este enfoque estandarizado permite una interacción eficiente y predecible mediante solicitudes HTTP, lo que facilita el desarrollo y la depuración.

**Flexibilidad y Modularidad:** La arquitectura se diseñó con un enfoque en la flexibilidad y la modularidad. Esto permite la fácil incorporación de nuevos módulos y funcionalidades, así como la adaptación a las cambiantes necesidades del laboratorio.

**Patrones de Diseño Aplicados:** Se aplicaron varios patrones de diseño, como el Modelo-Vista-Controlador (MVC), el patrón de API RESTful, el ORM y otros, para estructurar y optimizar el sistema de acuerdo con las mejores prácticas de desarrollo.

## Vista de Despliegue

La vista de despliegue describe cómo los componentes de la arquitectura de software se distribuirán e implementarán en un entorno real. se puede considerar el siguiente escenario de despliegue:

* **Front-end (Interfaz de Usuario):**

El front-end, construido con React se implementa en servidores locales.

* **Back-end (Lógica de Negocio):**

El back-end, construido con Express y Node.js, se desplegará en servidores locales.

* **Base de Datos (PostgreSQL):**

La base de datos PostgreSQL se desplegará en un servidor de bases de datos dedicado o en una instancia de base de datos gestionada localmente.

Se pueden establecer políticas de respaldo y recuperación para garantizar la integridad y disponibilidad de los datos críticos.

* **ORM Prisma:**

El ORM Prisma se integrará con la base de datos PostgreSQL y se desplegará junto con la aplicación back-end.

Prisma se configurará para conectarse a la instancia de base de datos y simplificar la manipulación de datos.

* **Medidas de Seguridad:**

Las medidas de seguridad, como los servidores de autenticación, cortafuegos y sistemas de detección de intrusiones se desplegarán en la capa de seguridad del entorno de despliegue, ya sea en el front-end o el back-end, según corresponda.

* **Sistemas de Respaldos y Recuperación:**

Los sistemas de respaldo y recuperación se configurarán para realizar copias de seguridad periódicas de la base de datos PostgreSQL y se desplegarán en un entorno de respaldo.

# Conclusiones

La arquitectura se enfoca en la mejora de procesos operativos y la eficiencia en la gestión de datos, lo que contribuye directamente a la calidad de los servicios en el laboratorio.

La separación de responsabilidades mediante un enfoque de capas facilita la mantenibilidad, escalabilidad y modularidad del sistema.

La elección de tecnologías compatibles, como React, Express, Node.js y Prisma, permite una construcción ágil y eficiente del sistema.

La implementación de medidas de seguridad en el front-end y el back-end protege los datos y garantiza el acceso autorizado, lo que es crucial para la confidencialidad y la integridad de la información.

La flexibilidad y modularidad en el diseño permiten la incorporación de nuevos módulos y funcionalidades, lo que asegura que el sistema pueda adaptarse a las cambiantes necesidades del laboratorio.

La implementación de una API RESTful facilita la comunicación eficiente entre los componentes del sistema, lo que mejora la experiencia del usuario y simplifica el desarrollo y la depuración.

La elección de PostgreSQL como base de datos y el uso de Prisma como ORM optimizan la gestión de datos y mejoran la eficiencia en la manipulación de información crítica.

Los sistemas de respaldo y recuperación de datos garantizan la disponibilidad continua de la información en caso de fallos o desastres, lo que es esencial para la continuidad del servicio.

# Referencias

Node.js. (s/f). Node.js. Recuperado el 21 de octubre de 2023, de https://nodejs.org/es

PostgreSQL. (2023, octubre 21). PostgreSQL. https://www.postgresql.org/

Prisma. (s/f). Prisma. Recuperado el 21 de octubre de 2023, de https://www.prisma.io/

React. (s/f). React.dev. Recuperado el 21 de octubre de 2023, de https://react.dev/