Trabajo
Fin de Grado

Generador de Tablas de Nucleidos

Alfonso Jesús Piñera Herrera

Ingeniería Informática

Este Trabajo de Fin de Grado desarrolla una herramienta para generar tablas de nucleidos parametrizadas en formato SVG, proporcionando una visualización clara y personalizable de los datos nucleares. Utiliza una biblioteca en Python para la lógica, una API en Flask para las peticiones y un frontend en React y Node.js para la interacción del usuario.

El proyecto se enfoca en ofrecer una solución flexible y robusta para la visualización de datos nucleares, integrando formatos de datos externos y facilitando su gestión. La aplicación de tecnologías modernas asegura una herramienta de alta calidad que cumple con necesidades académicas y prácticas.

Alfonso Jesús Piñera Herrera es un graduado en Ingenieria Informatica en la UGR nacido en Cieza, Murcia. Ha sido el encargado de desarrollar este proyecto.

Andrés María Roldán Aranda es el director académico del presente proyecto y el tutor del estudiante. Es profesor en el Departamento de Electrónica y Tecnología de Computadores en la Universidad de Granada.

UNIVERSIDAD DE GRANADA

Grado en Ingeniería Informática

Generador de Tablas de Nucleidos

Alfonso Jesús Piñera Herrera

2023/2024

Tutor: Andrés María Roldán Aranda

TRABAJO FIN DE GRADO

INGENIERÍA INFORMATICA

Generador de Tablas de Nucleidos

Autor

Alfonso Jesús Piñera Herrera (alumno)

Directores

Andrés María Roldán Aranda (tutor)

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, septiembre de 2024

Generador de tablas de nucleidos

Alfonso Jesus Pinera Herrera(alumno)

Palabras clave: Tabla de nucleótidos, desarrollo software, react, python, isótopos, flask, ...

Resumen

El objetivo principal de este TFG es desarrollar una herramienta para la generación de tablas de nucleidos parametrizadas en formato SVG, que permita una visualización clara y personalizable de los datos nucleares. Para ello, se ha diseñado una arquitectura que integra diversas tecnologías: una biblioteca en Python que se encarga de la lógica de generación de las tablas, una API desarrollada con Flask para gestionar las peticiones del frontend, y un frontend basado en React y Node.js que permite la interacción del usuario y la visualización en tiempo real.

La herramienta está pensada para ofrecer flexibilidad a los usuarios, permitiéndoles personalizar las tablas según diferentes parámetros y trabajar con datos externos en formato CSV o JSON. Además, se ha añadido soporte para la edición de configuraciones desde el frontend, facilitando una experiencia intuitiva y eficiente.

Project Title: Project Subtitle

Alfonso Jesús Pinera Herrera (student)

Keywords: Nuclide tables, software development, React, Python, isotopes, Flask, ...

Abstract

The main objective of this Final Degree Project (TFG) is to develop a tool for the generation of parametrized nuclide tables in SVG format, allowing for clear and customizable visualization of nuclear data. The project is built using an architecture that integrates various technologies: a Python library responsible for the logic of generating the tables, a Flask API to handle requests from the frontend, and a React/Node.js-based frontend that enables user interaction and real-time visualization.

The tool is designed to offer flexibility to users, allowing them to customize the tables based on different parameters and work with external data in CSV or JSON formats. Additionally, support for configuration editing from the frontend has been added, providing an intuitive and efficient user experience.

Yo, **Alfonso Jesus Pinera Herrera**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 48752658H, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Alfonso Jesus Pinera Herrera

Firmado por PIÑERA HERRERA ALFONSO JESUS - ***5265** el día 06/09/2024 con un certificado emitido por

Granada a 6 mes de 2024 .

D. **Andrés María Roldán Aranda (tutor)**, Profesor del Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Generador de Tablas de Tucleidos***, ha sido realizado bajo su supervisión por **Alfonso Jesus Pinera Herrera (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 6 de septiembre de 2024.

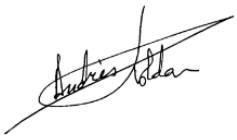


Figura 1: Firma del autor

Andrés María Roldán Aranda (tutor)

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mi familia, por su apoyo incondicional a lo largo de este camino académico. Su confianza y apoyo han sido mi mayor motor incluso en los momentos más difíciles.

A mis amigos, gracias por estar siempre ahí, tanto en los días de trabajo como en las noches de durums. Vuestra compañía han sido esenciales para mantenerme a flote y llegar hasta aquí.

Licencia

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este capítulo se presenta el contexto y la motivación del proyecto de desarrollo de un generador de tablas de nucleidos en formato SVG, su relevancia en el campo de la electrónica y tecnología de computadores, y los objetivos que se persiguen con su realización. Además, se describe brevemente la estructura del documento para guiar al lector a lo largo del trabajo.

1.1. Motivación

La tabla de nucleidos es una herramienta esencial en el estudio de la física nuclear y la química, proporcionando una representación bidimensional de los isótopos de los elementos, donde se muestran el número de protones (Z) y el número de neutrones (N) en los núcleos atómicos. Esta tabla permite a los investigadores y estudiantes obtener una visión clara de las propiedades nucleares y la estabilidad de los isótopos.

A pesar de la existencia de productos comerciales como la tabla de nucleidos Karlsruhe, estos suelen ser costosos y no siempre reflejan los datos más actualizados. Además, estas soluciones comerciales presentan limitaciones en cuanto a personalización y actualización de la información, lo que puede ser un obstáculo para instituciones educativas y de investigación que requieren información precisa y actualizada.

Por tanto, este proyecto tiene como objetivo desarrollar una herramienta accesible y flexible que permita la generación automática de tablas de nucleidos en formato SVG, utilizando datos actualizados. Este generador no solo será una alternativa más económica, sino que también ofrecerá una mayor capacidad de personalización, permitiendo a los usuarios ajustar la información presentada según sus necesidades específicas.

1.2. Objetivos y funcionalidades genéricas

El objetivo principal de este proyecto es crear una herramienta que permita la generación automática de tablas de nucleidos personalizadas en formato SVG, con la posibilidad de incluir información detallada y actualizada sobre cada nucleótido. Para lograr este objetivo, se han establecido los siguientes objetivos específicos:

- Desarrollar una librería en Python que permita el manejo de datos nucleares y la generación de gráficos en formato SVG.
- Implementar una API en Flask que permita a los usuarios cargar datos nucleares en formato CSV y configuraciones en formato JSON para generar tablas de nucleidos personalizadas.
- Crear una aplicación web que permita a los usuarios visualizar y modificar en tiempo real la tabla de nucleidos antes de descargarla en formato SVG.
- Asegurar que la herramienta soporte diferentes tamaños de salida (A1, A0) y permita escalado automático de la información en cada celda de la tabla.

1.3. Funcionalidad de la herramienta

La funcionalidad de la herramienta desarrollada en este proyecto se puede dividir en tres componentes principales:

1. **Librería en Python:** Una librería que maneja los datos nucleares y permite la creación de gráficos SVG. Esta librería debe ser capaz de leer datos nucleares desde un archivo CSV, procesar la información y generar una representación gráfica adecuada.
2. **API en Flask:** Un servidor web que permite a los usuarios subir archivos CSV con los datos de nucleidos y archivos JSON con configuraciones para personalizar la salida. La API generará un archivo SVG basado en estos datos y configuraciones, que los usuarios podrán descargar.
3. **Aplicación web:** Una interfaz de usuario que permite a los usuarios cargar datos, visualizar los cambios en la tabla de nucleidos en tiempo real, y descargar el archivo SVG final. Esta aplicación web debe ser intuitiva y ofrecer herramientas para ajustar la presentación de los datos en la tabla.

1.4. Estructura del proyecto

El proyecto está organizado en varios capítulos que cubren todos los aspectos del desarrollo del generador de tablas de nucleidos. A continuación, se describe brevemente el contenido de cada capítulo:

- **Capítulo 2: Especificación de requisitos** – Este capítulo detalla los requisitos funcionales y no funcionales del sistema, así como los perfiles de los usuarios y los casos de uso.
- **Capítulo 3: Análisis de Tecnologías** – Se presenta un análisis de las tecnologías existentes que son relevantes para la realización de un proyecto software como este.
- **Capítulo 4: Plan de Proyecto** – En este capitulo se detalla la organización de los recursos, la planificación temporal, el estudio de las tecnologías utilizadas y la estimación de costes.
- **Capítulo 5: Diseño** – Se describe el diseño propuesto para el sistema, la arquitectura que este tendrá, el prototipo de interfaz y los diagramas UML y una descripción detallada de los casos de uso del sistema.
- **Capítulo 6: Implementación** – En este capitulo se especifica el entorno de desarrollo usado, la estructura final usada, algunos ejemplos del código que interactúa con otras partes del sistema, el despliegue de la aplicación, y los desafíos encontrados durante la implementación del sistema.
- **Capítulo 7: Pruebas** – Aquí se cubrirán las distintas pruebas a las que el código es sometido para alcanzar los estándares de calidad propuestos.
- **Capítulo 8: Conclusiones** – En este capítulo se presentan las conclusiones del proyecto, incluyendo una reflexión sobre la experiencia personal y propuestas de mejora.

1.5. Publicación del proyecto

Todos los recursos generados en este proyecto pueden ser encontrados entre los siguientes enlaces:

1.5.1. Código:

Librería: https://git.granasat.space/alfonsojph786/nucleidechartlib.git **API**: https://git.granasat.space/alfonsojph786/nucleidechartAPI.git **Frontend**: https://git.granasat.space/alfonsojph786/nucleidechartfrontend.git

1.5.2. Paginas web:

Self-hosted(Servidor de desarrollo): nucleidechart.vaelico.es **Servidor AWS(Servidor de produccion)**: nc.vaelico.es **API**: nucleidechart.vaelico.es/API

Capítulo 2

Especificación de Requisitos

2.1. Descripción General del Sistema

El sistema consistirá en un entorno completo que permita a cualquier tipo de usuario generar tablas de nucleidos de forma simple e intuitiva, pudiendo adaptar la información mostrada a las necesidades del usuario.

2.2. Requisitos del Sistema

Una vez descrito el objetivo de este proyecto se ha realizado un análisis de las necesidades del sistema a desarrollar en base a la petición del cliente (tutor).

2.2.1. Requisitos Funcionales

- **RF1 - Carga de Archivos CSV y JSON**:
  - El sistema debe permitir al usuario cargar un archivo CSV que contiene los datos del conjunto de nucleidos de los que se quiere generar la tabla en formato SVG.
  - El sistema debe permitir al usuario cargar un archivo JSON que contiene las configuraciones de estilo del SVG de salida.
- **RF2 - Generación de SVG**:
  - El sistema debe generar un archivo SVG basado en los datos del archivo CSV y las configuraciones proporcionadas en el archivo JSON.
- **RF3 - Edición de Configuraciones SVG**:

- El sistema debe permitir al usuario modificar las configuraciones del SVG en tiempo real.
- Los cambios en las configuraciones deben reflejarse inmediatamente en el archivo SVG generado.

- **RF4 - Visualización e Interacción con SVG**:
  - El sistema debe permitir al usuario visualizar el SVG generado en el navegador.
  - El usuario debe poder interactuar con el SVG (e.g., zoom, pan, recarga) a través de la interfaz de usuario.
- **RF5 - Descarga de la tabla en formato SVG**:
  - El sistema debe permitir al usuario descargar SVG generado en el navegador.
  - El usuario debe poder a su vez poder descargar las configuraciones en formato JSON usadas para generar el SVG.

2.2.2. Requisitos No Funcionales

- **RF1 - Escalabilidad**:
  - El sistema debe ser capaz de manejar múltiples solicitudes de generación de SVG de manera concurrente.
- **RF2 - Usabilidad**:
  - La interfaz de usuario debe ser intuitiva y fácil de usar, facilitando la carga de archivos, la edición de configuraciones y la visualización de SVGs.
- **RF3 - Rendimiento**:
  - El sistema debe generar los archivos SVG de manera eficiente, minimizando el tiempo de procesamiento para proporcionar una experiencia de usuario fluida.

2.3. Criterios de validación

La validación de la aplicación se considerara respecto al grado de cumplimiento de los requisitos funcionales y no funcionales descritos anteriormente, para asegurar la robustez a través de todo el proyecto se realizaran pruebas unitarias en cada parte del sistema que garantizan la integridad de este.

Capítulo 3

Análisis del Sistema

Durante este capitulo analizare las distintas tecnologías que considero que podrían ser útiles para el desarrollo del sistema en su totalidad, partiendo de una arquitectura modular en la que agrupamos las distintas funcionalidades en distintos módulos dependiendo de su ámbito.

3.1. Tecnologías Disponibles

Pese a que hay una gran cantidad de aspectos a cubrir he considerado que las mas relevantes son las que cubren la estructura de backend, frontend y despliegue de la aplicación

3.1.1. Librería Generadora de Tablas de Nucleidos

Una parte fundamental en el proyecto sera una librería que sea la encargada de la lógica de la generación del SVG, siendo esta la base para el resto del sistema y permita seguir añadir funcionalidades en un futuro.

Lenguajes de Programación

En el desarrollo de aplicaciones para la generación y manipulación de gráficos vectoriales escalables (SVG), la elección del lenguaje de programación es crucial. Existen varios lenguajes que se destacan por su capacidad para manejar tareas de procesamiento de gráficos, cada uno con sus propias ventajas y desventajas. Entre otros, los mas prometedores para este proyecto fueron los siguientes:

- **Python**: Es uno de los lenguajes más populares en el desarrollo de aplicaciones de procesamiento de datos y gráficos. Python es conocido

por su simplicidad y legibilidad, lo que facilita el desarrollo rápido y el mantenimiento del código. Además, cuenta con una amplia variedad de librerías especializadas en la manipulación de gráficos, descritas mas adelante, que permiten la generación de SVG de manera eficiente y flexible. La principal desventaja de Python recae en que no es el lenguaje más rápido en términos de rendimiento bruto, lo que podría ser un inconveniente en aplicaciones que requieren una gran cantidad de procesamiento en tiempo real.

- **JavaScript**: Utilizado principalmente en el desarrollo web, JavaScript permite la manipulación de gráficos SVG directamente en el navegador. Esto es particularmente útil para aplicaciones interactivas que requieren la generación dinámica de gráficos. JavaScript ofrece potentes herramientas para la creación de visualizaciones de datos, pero la desventaja principal es que al ser un lenguaje interpretado, puede tener limitaciones de rendimiento en comparación con lenguajes compilados.

- **Ruby**: Ruby es conocido por su elegancia y simplicidad en el desarrollo de aplicaciones web. Aunque no es tan popular como Python o JavaScript para la generación de gráficos, también ofrece librerías como *RMagick* que pueden ser utilizadas para manipular gráficos vectoriales. Sin embargo, la comunidad de desarrollo en torno a SVG en Ruby es más limitada, lo que puede dificultar encontrar soporte o librerías especializadas.

Aspecto	**Python**	**JavaScript**	**Ruby**
Ventajas	Simplicidad, gran comunidad, librerías avanzadas para gráficos (svgwrite)	Manipulación directa en el navegador, alta interactividad	Código elegante y legible
Desventajas	Menor rendimiento en tiempo real	Limitado en procesamiento gráfico avanzado	Comunidad más pequeña en gráficos SVG, menos librerías disponibles
Rendimiento	Medio	Medio-alto	Medio-bajo
Facilidad de Uso	Alta	Alta	Alta
Compatibilidad	Compatible con librerías de generación de SVG	Muy compatible en entornos web	Menos compatible para gráficos complejos
Escalabilidad	Moderada	Alta	Baja

Cuadro 3.1: Comparación de Lenguajes de Programación para la Librería Generadora de Tablas de Nucleidos

Por su simplicidad, el conocimiento previo que tengo sobre este y la variedad de librerías existentes para la generación de SVG he decidido realizar este módulo en Python, si bien el rendimiento que puede tener podría ser un problema, la estimación de la cantidad de procesos concurrentes que podría llegar a tener este sistema no es tan alta, asi que sus ventajas opacan las desventajas que este podría tener.

Librerías de Generación de SVG

La elección de la librería adecuada para la generación de SVG es fundamental para garantizar que la aplicación cumpla con los requisitos de funcionalidad y rendimiento. A continuación, se presentan algunas de las principales librerías disponibles en el ecosistema de Python:

- **svgwrite**: Es una librería de Python diseñada para la creación y manipulación de archivos SVG. Se destaca por su simplicidad y facilidad de uso, proporcionando una API intuitiva que permite definir y estilizar elementos SVG de manera programática. *svgwrite* es especialmente

útil para proyectos que requieren la generación de gráficos SVG personalizados sin la necesidad de conocimientos avanzados en gráficos. Su desventaja es que puede carecer de algunas características avanzadas que ofrecen otras librerías más robustas.

- **CairoSVG**: Esta librería es una herramienta más robusta para la creación y manipulación de SVG, con capacidades adicionales para la conversión de SVG a otros formatos, como PNG o PDF. *CairoSVG* es adecuada para aplicaciones que requieren gráficos más complejos o que necesitan integrarse con otras herramientas de procesamiento de imágenes. Sin embargo, su uso puede ser más complicado y menos intuitivo que *svgwrite* para tareas básicas de generación de SVG.

- **ReportLab**: Aunque es más conocida por su capacidad para generar documentos PDF, *ReportLab* también ofrece soporte para gráficos SVG. Es una opción potente si se requiere combinar gráficos vectoriales con generación de documentos en otros formatos. No obstante, para aplicaciones que se centran exclusivamente en SVG, puede ser excesiva y menos eficiente que opciones como *svgwrite* o *CairoSVG*.

Aspecto	 **svgwrite**	 **CairoSVG**	 **Reportlab**
Ventajas	Simple, API intuitiva	Soporta conversión a otros formatos, manejo de gráficos complejos	Potente para gráficos y documentos PDF/SVG combinados
Desventajas	Carece de características avanzadas	Mayor complejidad en su uso	Menos eficiente para proyectos centrados exclusivamente en SVG
Rendimiento	Alto	Alto	Medio
Facilidad de Uso	Alta	Media	Baja
Compatibilidad	Compatible con proyectos simples y medios	Altamente compatible con gráficos complejos	Compatible con proyectos grandes y PDF/SVG
Escalabilidad	Moderada	Alta	Alta

Cuadro 3.2: Comparación de Librerías de Generación de SVG

Debido a su simpleza y rendimiento he decidido usar svgwrite, ya que ofrece la funcionalidad necesaria para realizar este proyecto sin sacrificar rendimiento ni legibilidad.

3.1.2. Backend

Para realizar el backend he decidido hacer una API que pueda hacer uso de la librería previamente descrita en el apartado anterior.

Tecnologías de Backend

El desarrollo del backend de una aplicación web requiere la selección cuidadosa de las tecnologías y frameworks que mejor se adapten a los requisitos específicos del proyecto. Entre las opciones disponibles en el mercado, se destacan:

Además, dentro de estos lenguajes podemos encontrar una gran variedad de frameworks que simplifican y aceleran la implementación del sistema.

- **Flask (Python)**: Flask es un microframework para Python que es muy popular debido a su simplicidad y flexibilidad. Permite a los desarrolladores crear aplicaciones web y APIs con un mínimo de código y configuración. Flask es ideal para proyectos que no requieren la complejidad de un framework completo como Django. Su ecosistema de extensiones facilita la adición de funcionalidades adicionales según sea necesario. Sin embargo, su simplicidad también significa que algunas funcionalidades comunes deben ser implementadas manualmente o a través de extensiones, lo que podría aumentar la carga de desarrollo en proyectos más grandes.
- **Django (Python)**: Django es un framework web completo y de alto nivel que promueve el desarrollo rápido y un diseño limpio y pragmático. Ofrece una gran cantidad de funcionalidades integradas, como un ORM robusto, un sistema de autenticación, y herramientas de administración. Django es ideal para aplicaciones más grandes y complejas que requieren una estructura bien definida desde el principio. No obstante, su complejidad y la curva de aprendizaje pueden ser excesivos para proyectos más pequeños o menos complejos.
- **Express.js (Node.js)**: Express.js es un framework web minimalista para Node.js que se utiliza para construir aplicaciones web y APIs. Es extremadamente flexible y permite a los desarrolladores definir cómo gestionar las solicitudes HTTP en una aplicación. Express.js es muy popular en el ecosistema de JavaScript debido a su simplicidad y la

facilidad con la que se integra con otras herramientas y bibliotecas de Node.js. Sin embargo, al igual que Flask, su minimalismo significa que los desarrolladores deben implementar manualmente muchas funcionalidades que otros frameworks podrían proporcionar de manera predeterminada.

Aspecto	 **Flask**	**Django**	 **express**
Ventajas	Microframework flexible y simple de usar	Framework completo, con muchas herramientas integradas	Minimalista, rápido para manejar operaciones de I/O asíncronas
Desventajas	Algunas funcionalidades deben implementarse manualmente	Mayor complejidad, curva de aprendizaje más alta	Se requieren bibliotecas adicionales para funcionalidades comunes
Rendimiento	Alto	Medio	Alto
Facilidad de Uso	Alta	Media	Alta
Compatibilidad	Compatible con muchos módulos de Python	Ideal para aplicaciones más complejas	Integración fluida con herramientas de JavaScript
Escalabilidad	Alta	Alta	Alta

Cuadro 3.3: Comparación de Tecnologías de Backend

Una vez mas he decidido hacer uso de Python y Flask por su simplicidad y por ser una herramienta con la que he trabajado previamente, aunque destaque por su simplicidad, es mas que suficiente para manejar las peticiones de generación de los SVG haciendo uso de la libreria de forma eficiente.

3.1.3. Frontend

Pese a que existen distintas herramientas como Flutter (un framework de Dart mantenido por Google que permite la creación de interfaces en multiplataforma), Electron o Kotlin (framework de java que permite la creación de aplicaciones moviles) que facilitan la creación de interfaces gráficas y el uso de llamadas a la API, he decidido que el frontend sea una interfaz web, debido a su simplicidad y portabilidad.

Lenguajes de Programación

Para el desarrollo frontend, se consideraron varios lenguajes que son comunes en la creación de interfaces de usuario web. Los lenguajes evaluados incluyen:

- **JavaScript**: Es el lenguaje estándar para el desarrollo frontend en la web. JavaScript es compatible con todos los navegadores modernos y es esencial para la creación de aplicaciones web interactivas. Permite manipular el DOM, manejar eventos, y realizar solicitudes asíncronas al servidor. Además, la mayoría de los frameworks y librerías frontend, como React, Angular y Vue.js, están basados en JavaScript. Su principal desventaja es que, al ser interpretado, puede ser menos eficiente en comparación con lenguajes compilados.
- **TypeScript**: Es un superconjunto de JavaScript que añade tipado estático al lenguaje. TypeScript mejora la robustez del código y facilita el mantenimiento de proyectos grandes, permitiendo a los desarrolladores detectar errores en tiempo de compilación. Aunque introduce una fase adicional de compilación, lo que puede alargar el ciclo de desarrollo, su integración con herramientas como React es muy fluida, y es especialmente beneficioso en proyectos de gran escala que requieren un alto grado de fiabilidad.
- **HTML/CSS**: Aunque no son lenguajes de programación en el sentido estricto, HTML y CSS son fundamentales para el desarrollo frontend. HTML estructura el contenido de las páginas web, mientras que CSS se encarga de la presentación visual. Junto con JavaScript, forman la base de todas las aplicaciones web frontend. La principal limitación es que, por sí solos, no son suficientes para crear aplicaciones dinámicas e interactivas, lo que requiere la integración con JavaScript o TypeScript.

Tecnologías de Frontend

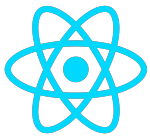
Además del lenguaje de programación, la elección de tecnologías y frameworks es esencial para construir una interfaz de usuario eficiente y atractiva. Entre las opciones evaluadas se incluyen:

- **React**: Es una biblioteca de JavaScript para construir interfaces de usuario. React permite crear componentes reutilizables que manejan su propio estado, lo que facilita el desarrollo de interfaces complejas y dinámicas. Es altamente popular y cuenta con un gran ecosistema de herramientas y librerías complementarias. Sin embargo, React se centra en la vista de la aplicación, por lo que a menudo es necesario

combinarlo con otras tecnologías para construir una aplicación completa.

- **Angular**: Es un framework de desarrollo web completo mantenido por Google. Angular es más descriptivo que React y ofrece una solución integral para el desarrollo frontend, incluyendo gestión de estado, enrutamiento, y herramientas de desarrollo. Es adecuado para proyectos grandes que se benefician de una arquitectura bien definida, aunque su curva de aprendizaje puede ser empinada.

- **Vue.js**: Es un framework progresivo de JavaScript que se ha ganado popularidad por su simplicidad y flexibilidad. Vue.js es similar a React en cuanto a que permite la creación de componentes reutilizables, pero es más fácil de aprender y configurar. Es una excelente opción para proyectos de cualquier tamaño, aunque no cuenta con el mismo nivel de adopción empresarial que React o Angular.

Aspecto			
Ventajas	Modularización, alta capacidad para manejar estados dinámicos	Framework completo con muchas herramientas integradas	Simplicidad y flexibilidad, ideal para proyectos pequeños y medianos
Desventajas	Necesita otras herramientas para un desarrollo completo	Curva de aprendizaje más alta	Menos adopción en entornos empresariales
Rendimiento	Alto	Medio	Alto
Facilidad de Uso	Alta	Media	Alta
Compatibilidad	Compatible con muchas herramientas de JS	Ideal para aplicaciones grandes	Compatible con proyectos de cualquier tamaño
Escalabilidad	Alta	Alta	Alta

Cuadro 3.4: Comparación de Tecnologías de Frontend

3.2. Desarrollo de la interfaz

Para el diseño inicial se realizaran unos mockups que servirán para delinear una serie de pautas a seguir respecto al diseño final de la aplicación web, sin embargo estos mockups no son una representación exacta del diseño final, el cual puede ser modificado para poder adaptarse a las necesidades de la aplicación.

3.2.1. Diseño Preliminar

Para el diseño preliminar de la interfaz, se han considerado las siguientes herramientas:

- **Canva:** Canva es una herramienta de diseño gráfico en línea que permite crear prototipos y mockups de manera rápida y fácil. Ofrece una amplia gama de plantillas y herramientas de diseño que facilitan la creación de representaciones visuales de la interfaz de usuario. Aunque es menos específica para el diseño de interfaces web interactivas, es adecuada para la creación de prototipos iniciales.

- **Adobe XD:** Adobe XD es una herramienta profesional para el diseño de interfaces y experiencias de usuario. Permite crear prototipos interactivos y realizar pruebas de usabilidad. Su integración con otras herramientas de Adobe y su enfoque en el diseño de interfaces web y móviles lo hacen ideal para proyectos más detallados y complejos.

- **Figma:** Figma es una herramienta de diseño colaborativo basada en la web que permite a los equipos trabajar juntos en tiempo real. Ofrece potentes capacidades para crear prototipos interactivos y maquetas detalladas. Es especialmente útil para proyectos que requieren una colaboración estrecha entre diseñadores y desarrolladores.

Para este proyecto, se ha optado por *Canva* para los mockups iniciales debido a su facilidad de uso y la rapidez con la que se pueden generar representaciones visuales.

Finalmente la arquitectura del sistema quedaría dividida en dos módulos principales distintos: backend (formado por la API y la librería) y frontend, ambos necesarios para el correcto funcionamiento del sistema, por lo que he decidido desarrollar una forma de realizar el despliegue de estos dos módulos de forma automatizada para que su despliegue sea garantizado independientemente del sistema en el que se use.

3.2.2. Tecnologías de Despliegue

- **Docker:** Docker es una plataforma que permite empaquetar aplicaciones en contenedores, asegurando que el software funcione de manera consistente en cualquier entorno. Facilita la creación de imágenes de contenedor para cada componente del sistema, incluyendo el backend y el frontend. Además, Docker simplifica la integración continua y el despliegue, proporcionando una solución robusta y flexible para la gestión de dependencias y la configuración del entorno de ejecución.
- **Kubernetes:** Kubernetes es una plataforma de orquestación de contenedores que automatiza el despliegue, la escalabilidad y la gestión de aplicaciones en contenedores. Ofrece características avanzadas como balanceo de carga, recuperación ante fallos y despliegue de versiones. Aunque proporciona un control más fino y una mayor capacidad de escalabilidad, su complejidad puede ser excesiva para proyectos más pequeños o menos complejos.
- **Vagrant:** Vagrant es una herramienta que facilita la creación y configuración de entornos de desarrollo virtualizados. Permite definir y gestionar entornos de desarrollo de manera consistente mediante el uso de máquinas virtuales. Aunque es útil para entornos de desarrollo y pruebas, puede ser menos adecuado para el despliegue de producción en comparación con Docker y Kubernetes.

Para este proyecto, se ha decidido utilizar *Docker* para el despliegue, debido a su simplicidad en la gestión de contenedores y su compatibilidad con las herramientas de integración continua y despliegue automático. Además, Docker ofrece una solución efectiva para manejar tanto el backend como el frontend de manera independiente.

3.3. Conclusiones del Análisis

En base al análisis de las tecnologías disponibles, se realizaron las siguientes elecciones para el desarrollo del sistema:

- **Librería de Generación de SVG:** Se seleccionó *svgwrite* por su simplicidad y facilidad de integración con Python, lo que permite una implementación rápida y eficiente para la generación de gráficos SVG personalizados.
- **Backend:** Se eligió *Flask* como tecnología de backend debido a su flexibilidad y compatibilidad con Python, lo que facilita el desarrollo de la API y la integración con la lógica de generación de SVG.

Aspecto	**Docker**	**Kubernetes**	**Vagrant**
Ventajas	Empaquetado consistente, gestión fácil de dependencias	Orquestación avanzada, automatiza la escalabilidad y gestión de fallos	Consistencia en entornos de desarrollo
Desventajas	Requiere conocimientos básicos de contenedores	Complejidad en su configuración y mantenimiento	Menos adecuado para despliegue en producción
Rendimiento	Alto	Alto	Medio
Facilidad de Uso	Alta	Baja	Media
Compatibilidad	Altamente compatible con CI/CD	Ideal para aplicaciones grandes	Ideal para entornos de desarrollo
Escalabilidad	Alta	Muy alta	Media

Cuadro 3.5: Comparación de Tecnologías de Despliegue

- **Lenguaje de Programación para el Backend:** Se optó por *Python*, dado su ecosistema robusto y la compatibilidad con las librerías seleccionadas para la manipulación de gráficos.
- **Frontend:** Se decidió utilizar *JavaScript* junto con *React* para el desarrollo frontend, aprovechando su flexibilidad, la posibilidad de crear componentes reutilizables, y el amplio soporte de la comunidad.

Capítulo 4

Plan de Proyecto

En este capitulo realizare una planificación de todos los aspectos necesarios para poder realizar la implementación y despliegue de la aplicación.

4.1. Organización y Recursos

En este capítulo se detallan los recursos humanos y materiales necesarios para el desarrollo del proyecto. Además, se presentan las estimaciones de los costes asociados a cada recurso, lo que permite una planificación financiera más precisa.

4.1.1. Estructura Organizativa

La estructura organizativa del proyecto se basa en un enfoque ágil, con un equipo multidisciplinario que trabaja en estrecha colaboración para lograr los objetivos del proyecto. A continuación, se detallan los roles clave en el equipo de desarrollo.

4.1.2. Recursos Humanos

Los recursos humanos son fundamentales para el éxito del proyecto. A continuación, se describen las posiciones clave necesarias, junto con un ejemplo de salario medio para cada posición en España.

- **Desarrollador Frontend:** Encargado de implementar la interfaz de usuario utilizando las tecnologías seleccionadas.
  Salario medio: 30.000 - 35.000 € anuales.

- **Desarrollador Backend:** Responsable del desarrollo de la lógica del servidor, bases de datos y API.
  Salario medio: 35.000 - 40.000 € anuales.

- **DevOps:** Especialista en la integración y despliegue continuo, así como en la gestión de la infraestructura en la nube.
  Salario medio: 40.000 - 45.000 € anuales.

- **Gestor de Proyecto:** Responsable de coordinar el proyecto, asegurando que se cumplan los plazos y el presupuesto.
  Salario medio: 45.000 - 50.000 € anuales.

4.1.3. Recursos Materiales

Los recursos materiales comprenden tanto el hardware como el software necesario para el desarrollo y ejecución del proyecto. A continuación, se detallan estos recursos junto con una estimación de los costes asociados.

Hardware

Recurso	Descripción	Coste Estimado (mensual)
Servidor EC2	Instancia t2.medium para desarrollo y pruebas	50 €
Almacenamiento S3	Almacenamiento de datos para backup	20 €

Cuadro 4.1: Costes estimados de servidores en la nube

Recurso	Descripción	Coste Estimado (unidad)
Dell latitude	Portátil de desarrollo con 16 GB de RAM	900 €

Cuadro 4.2: Costes estimados de equipos de desarrollo

Software

Recurso	Descripción	Coste Estimado
Windows 11	Incluido con los equipos de desarrollo	Incluido
Ubuntu	Sistema operativo de código abierto	Gratuito

Cuadro 4.3: Costes estimados de sistemas operativos

Recurso	Descripción	Coste Estimado (anual)
JetBrains All Products Pack	Acceso a todos los productos JetBrains	649 €

Cuadro 4.4: Costes estimados de IDE

Recurso	Descripción	Coste Estimado (anual)
ClickUp	Gestión de tareas y proyectos	96 € por usuario

Cuadro 4.5: Costes estimados de herramientas de gestión de proyectos

Recurso	Descripción	Coste Estimado (mensual)
Docker	Versión gratuita para la mayoría de los casos de uso	Gratuito

Cuadro 4.6: Costes estimados de herramientas de despliegue

4.2. Planificación

La planificación del proyecto se basa en un enfoque ágil, con iteraciones cortas y entregas incrementales. A continuación, se presenta el plan de proyecto detallado, que incluye las tareas, los plazos y las dependencias entre ellas.

4.2.1. Resumen de Funcionalidad

A continuación se presenta un resumen de las funcionalidad a desarrollar, ya descritas previamente en el punto 2.2.

RF	Resumen de Funcionalidad	Prioridad
RF1	Permite al usuario cargar un archivo CSV con los datos del conjunto de nucleidos.	5
RF1	Permite al usuario cargar un archivo JSON con las configuraciones de estilo del SVG.	5
RF2	Genera un archivo SVG basado en los datos del archivo CSV.	5
RF2	Genera un archivo SVG basado en las configuraciones proporcionadas en el archivo JSON.	5
RF3	Permite al usuario modificar las configuraciones del SVG en tiempo real.	4
RF3	Los cambios en las configuraciones deben reflejarse inmediatamente en el archivo SVG generado.	4
RF4	Permite al usuario visualizar el SVG generado en el navegador.	5
RF4	Permite al usuario interactuar con el SVG (e.g., zoom, pan, recarga).	4
RF5	Permite al usuario descargar el archivo SVG generado en el navegador.	5
RF5	Permite al usuario descargar el archivo JSON con las configuraciones usadas para generar el SVG.	5
RF1	El sistema debe manejar múltiples solicitudes de generación de SVG de manera concurrente.	3
RF2	La interfaz de usuario debe ser intuitiva y fácil de usar, facilitando la carga de archivos y la edición de configuraciones.	4
RF3	El sistema debe generar los archivos SVG de manera eficiente, minimizando el tiempo de procesamiento.	4

Cuadro 4.7: Resumen de Funcionalidad de la Aplicación

4.2.2. División de Tareas por Sprint

Sprint 1: Desarrollo de la Librería en Python

El primer sprint se centra en la creación de una librería en Python capaz de generar archivos SVG a partir de archivos CSV y JSON, así como en la implementación de pruebas unitarias.

Tarea	Descripción	Duración
1.1	Desarrollo de la librería en Python	1 semana
1.2	Integración con archivos CSV	1 semana
1.3	Integración con archivos JSON	1 semana
1.4	Generación de SVG	1 semana
1.5	Pruebas Unitarias	1 semana

Cuadro 4.8: Tareas del Sprint 1: Desarrollo de la Librería en Python

Sprint 2: Desarrollo de la API en Flask

El segundo sprint se dedica a la creación de la API en Flask que utiliza la librería desarrollada en el primer sprint, así como al despliegue inicial y pruebas unitarias.

Tarea	Descripción	Duración
2.1	Desarrollo de la API en Flask	1 semana
2.2	Implementación de Endpoints	1 semana
2.3	Despliegue inicial de la API	1 semana
2.4	Pruebas Unitarias y de Integración	1 semana

Cuadro 4.9: Tareas del Sprint 2: Desarrollo de la API en Flask

Sprint 3: Desarrollo de la Aplicación en Node.js y React

El tercer sprint se enfoca en el desarrollo de la aplicación frontend en Node.js y React, que utiliza la API y la librería para generar y visualizar SVGs, y permitir la modificación de configuraciones.

Tarea	Descripción	Duración
3.1	Configuración del proyecto React	1 semana
3.2	Implementación de carga de archivos	1 semana
3.3	Integración con la API	1 semana
3.4	Implementación de visualización de SVG	1 semana
3.5	Implementación de edición de configuraciones	1 semana
3.6	Pruebas Unitarias y de Integración	1 semana

Cuadro 4.10: Tareas del Sprint 3: Desarrollo de la Aplicación en Node.js y React

4.2.3. Diagrama de Gantt

4.3. Presupuesto

Finalmente, se presenta un presupuesto estimado para el desarrollo del proyecto, que incluye los costes de los recursos humanos y materiales necesarios.

Detalle	Un.	Precio/Un.	Subtotal	Total
Recursos Humanos				
Desarrollador Frontend	1	32.500 €	32.500 €	32.500 €
Desarrollador Backend	1	37.500 €	37.500 €	37.500 €
DevOps	1	42.500 €	42.500 €	42.500 €
Gestor de Proyecto	1	47.500 €	47.500 €	47.500 €
Hardware				
Servidor EC2	12	50 €	600 €	600 €
Almacenamiento S3	12	20 €	240 €	240 €
Portátiles de desarrollo	4	900 €	3.600 €	3.600 €
Software				
JetBrains All Products Pack	1	649 €	649 €	649 €
ClickUp	1	96 €	96 €	96 €
Subtotal				165.685 €
Impuestos (21 %)				34.688,85 €
Beneficio Industrial (5 %)				7.259,25 €
Total				207.633,10 €

Capítulo 5

Diseño

5.1. Arquitectura del Sistema

La arquitectura del sistema es modular y está basada en componentes, permitiendo la escalabilidad y mantenibilidad del sistema. Se sigue un enfoque de diseño en capas, donde cada capa tiene una responsabilidad específica dentro del sistema.

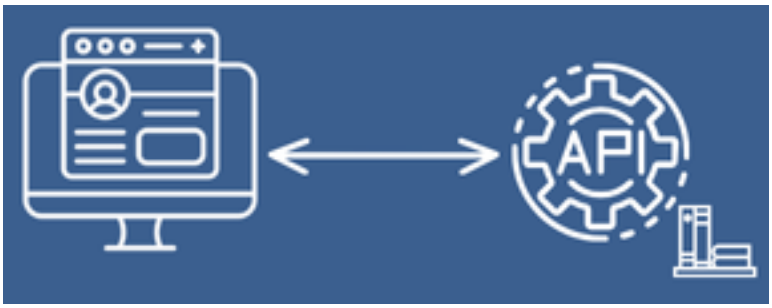


Figura 5.1: Arquitectura modular del sistema

5.1.1. Capa de Presentación(Frontend)

La API será la encargada de procesar las solicitudes del frontend. Recibe los archivos subidos y, tras procesarlos, devuelve el SVG generado. Actúa como intermediario entre el frontend y la lógica de negocio, asegurándose de gestionar las solicitudes y devolver las respuestas correctamente.

5.1.2. Capa de Aplicación (API)

La capa de aplicación contiene la lógica principal del sistema. Esta capa gestiona las reglas de negocio, los flujos de trabajo y las interacciones entre los diferentes módulos del sistema. Esta parte está comprendida dentro de la API en flask, que es la parte encargada de generar las tablas en formato SVG.

5.1.3. Capa de Lógica de Negocio (Librería

La capa de integración maneja la comunicación entre la aplicación y servicios externos. Esta capa asegura que las interacciones externas sean gestionadas de manera eficiente y segura. La capa de integración está incorporada como un servicio dentro del frontend, donde se encarga de realizar las peticiones solicitadas por los componentes a la API y manejar las respuestas de la API.

5.2. Diseño Modular

El diseño del sistema sigue un enfoque modular, donde cada módulo se desarrolla de manera independiente pero puede interactuar con otros módulos a través de interfaces bien definidas.

5.2.1. Módulo de Presentación (Frontend en React)

- Interacción con el usuario.
- Permite la carga de archivos (CSV y JSON).
- Modifica en tiempo real las configuraciones de la tabla de nucleidos.
- Envía solicitudes a la API y recibe el SVG generado para mostrarlo al usuario.

5.2.2. Módulo de API (Flask)

- Recibe peticiones del frontend.
- Procesa archivos subidos (CSV y JSON).
- Envía los datos al módulo de lógica de negocio para generar el SVG.
- Devuelve el SVG al frontend.

5.2.3. Módulo de Lógica de Negocio (Librería de Generación de SVG)

- Genera tablas o cajas individuales de nucleidos a partir de un vector.
- Recibe listas de nucleidos desde la API y genera el SVG.
- Acepta configuraciones personalizadas para la generación de las tablas.

5.2.4. Módulo de Gestión de Datos

- Extrae listas de nucleidos desde archivos CSV.
- Prepara los datos para ser utilizados por el módulo de generación de SVG.

5.3. Interfaz de Usuario

El diseño de la interfaz de usuario se centró en la estabilidad, con un enfoque en la simplicidad y la eficiencia. Se desarrollaron interfaces intuitivas que permiten al usuario interactuar con el sistema de manera efectiva.

5.3.1. Principios de Diseño

El diseño de la interfaz de usuario se basó en los siguientes principios:

- **Simplicidad:** Se priorizó un diseño simple y directo para minimizar la curva de aprendizaje.
- **Accesibilidad:** La interfaz fue diseñada para ser accesible a todos los usuarios, incluyendo aquellos con discapacidades.
- **Consistencia:** Se mantuvo una coherencia en el diseño visual y funcional a lo largo de todas las pantallas del sistema.

5.3.2. Prototipos de Interfaz

Se desarrollaron prototipos de interfaz para visualizar y validar el diseño antes de su implementación. Estos prototipos permitieron realizar ajustes basados en la retroalimentación de los usuarios.

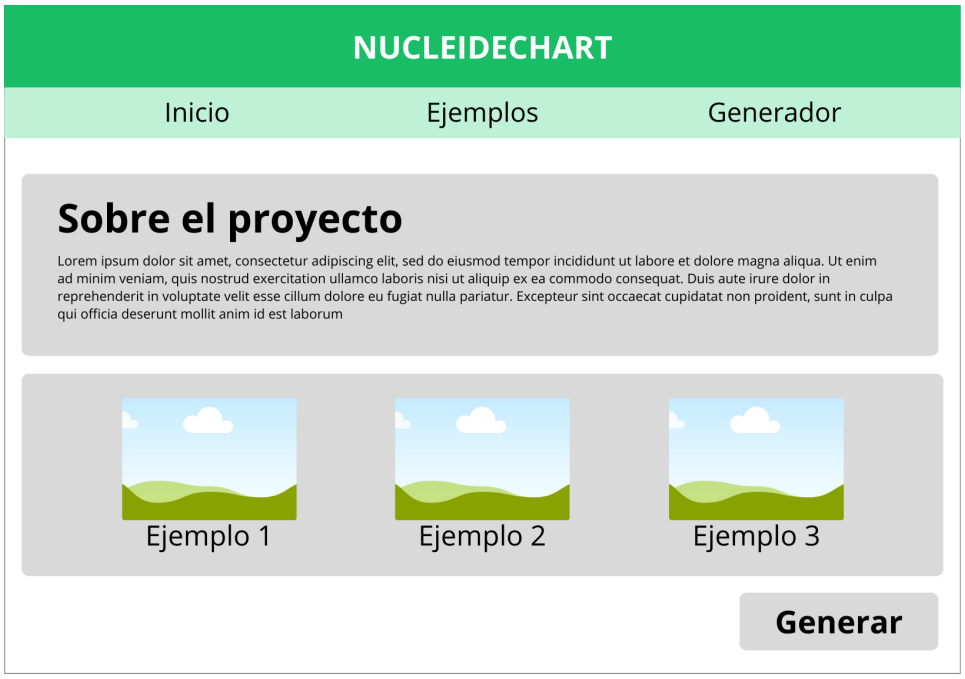


Figura 5.2: Prototipo de la interfaz de inicio

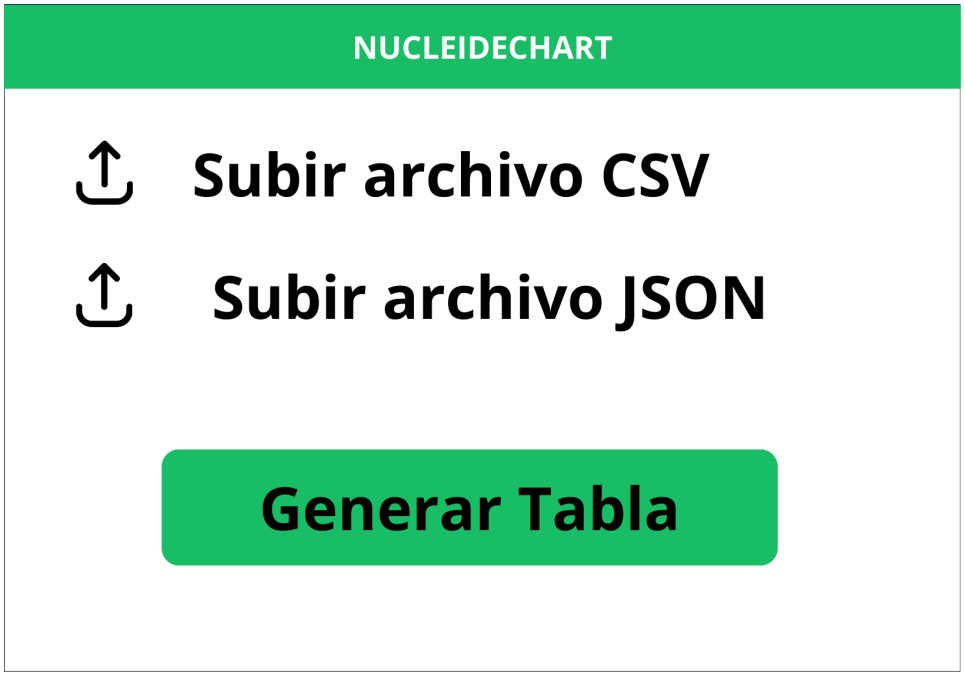


Figura 5.3: Prototipo de la interfaz del generador

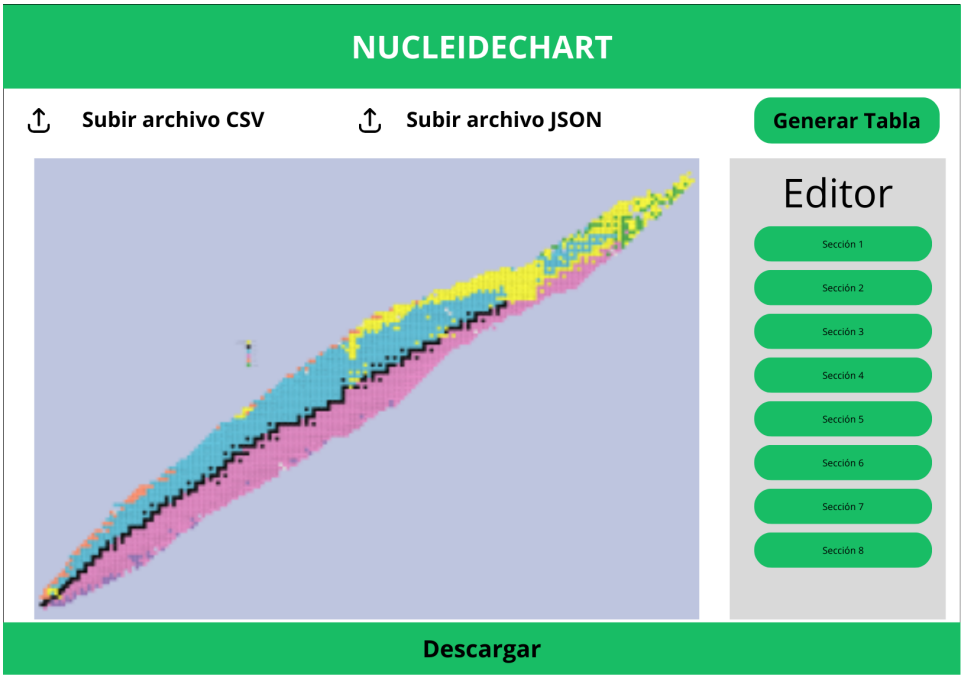


Figura 5.4: Prototipo de la interfaz del editor

5.4. Casos de Uso

5.4.1. Descripción de los Casos de Uso

Caso de Uso	Cargar Archivo CSV		**ID**	CU-01
Actor	Usuario			
Tipo	Primario			
Referencias	Ninguna			
Precondición	El usuario tiene acceso a un archivo CSV válido.			
Postcondición	El archivo CSV es aceptado y está listo para ser procesado.			
Autor	Alfonso Piñera Herrera	**Fecha**	21-08-24	**Versión** 1.0
Propósito	Permitir al usuario cargar un archivo CSV que contiene los datos para generar un archivo SVG.			
Resumen	El usuario selecciona y carga un archivo CSV desde su dispositivo. El sistema valida el formato y lo envía al backend para su procesamiento.			

Paso	Descripción
Paso 1	El usuario selecciona el archivo CSV desde su dispositivo.
Paso 2	El sistema valida el formato del archivo.
Paso 3	El archivo es enviado al backend para su procesamiento.

Cuadro 5.1: Descripción del Caso de Uso: Cargar Archivo CSV

Caso de Uso	Cargar Archivo JSON			**ID**	CU-02
Actor	Usuario				
Tipo	Primario				
Referencias	CU-01				
Precondición	El usuario tiene acceso a un archivo JSON válido.				
Postcondición	El archivo JSON es aceptado y está listo para ser procesado.				
Autor	Alfonso Piñera Herrera	**Fecha**	21-08-24	**Versión**	1.0
Propósito	Permitir al usuario cargar un archivo JSON que contiene las configuraciones para la generación de un archivo SVG.				
Resumen	El usuario selecciona y carga un archivo JSON desde su dispositivo. El sistema valida el formato y lo envía al backend para su procesamiento.				

Paso	Descripción
Paso 1	El usuario selecciona el archivo JSON desde su dispositivo.
Paso 2	El sistema valida el formato del archivo.
Paso 3	El archivo es enviado al backend para su procesamiento.

Cuadro 5.2: Descripción del Caso de Uso: Cargar Archivo JSON

Caso de Uso	Generar SVG			**ID**	CU-03
Actor	Sistema				
Tipo	Secundario				
Referencias	CU-01, CU-02				
Precondición	Los archivos CSV y JSON han sido cargados y validados.				
Postcondición	El archivo SVG es generado y enviado al frontend para su visualización.				
Autor	Alfonso Piñera Herrera	**Fecha**	21-08-24	**Versión**	1.0
Propósito	Generar un archivo SVG basado en los datos del archivo CSV y las configuraciones del archivo JSON.				
Resumen	El sistema procesa los datos del archivo CSV y aplica las configuraciones del archivo JSON para generar un archivo SVG.				

Paso	Descripción
Paso 1	El sistema procesa los datos del archivo CSV.
Paso 2	El sistema aplica las configuraciones del archivo JSON.
Paso 3	El sistema genera el archivo SVG.

Cuadro 5.3: Descripción del Caso de Uso: Generar SVG

Caso de Uso	Editar Configuracion del SVG			**ID**	CU-04
Actor	Usuario				
Tipo	Primario				
Referencias	CU-03				
Precondición	El archivo SVG ha sido generado.				
Postcondición	El archivo SVG se actualiza con las nuevas configuraciones.				
Autor	Alfonso Piñera Herrera	**Fecha**	21-08-24	**Versión**	1.0
Propósito	Permitir al usuario modificar las configuraciones del archivo SVG y ver los cambios en tiempo real.				
Resumen	El usuario accede a las opciones de configuración del archivo SVG, realiza las modificaciones necesarias y observa los cambios reflejados en el gráfico generado.				

Paso	**Descripción**
Paso 1	El usuario selecciona la opción para editar el archivo SVG.
Paso 2	El usuario modifica las configuraciones deseadas (colores, tamaños, etiquetas, etc.).
Paso 3	El sistema aplica los cambios al archivo SVG y lo actualiza en tiempo real.

Cuadro 5.4: Descripción del Caso de Uso: Editar Configuracion del SVG

Caso de Uso	Guardar Archivo SVG			**ID**	CU-05
Actor	Usuario				
Tipo	Primario				
Referencias	CU-03, CU-04				
Precondición	El archivo SVG ha sido generado y/o editado.				
Postcondición	El archivo SVG es guardado en el dispositivo del usuario.				
Autor	Alfonso Piñera Herrera	**Fecha**	21-08-24	**Versión**	1.0
Propósito	Permitir al usuario guardar el archivo SVG generado o editado en su dispositivo para su posterior uso.				
Resumen	El usuario selecciona la opción de guardar, y el sistema guarda el archivo SVG en la ubicación seleccionada por el usuario en su dispositivo.				

Paso	**Descripción**
Paso 1	El usuario selecciona la opción de guardar el archivo SVG.
Paso 2	El usuario elige la ubicación y el nombre del archivo a guardar.
Paso 3	El sistema guarda el archivo SVG en la ubicación seleccionada.

Cuadro 5.5: Descripción del Caso de Uso: Guardar Archivo SVG

Caso de Uso	Visualizar Archivo SVG			**ID**	CU-06
Actor	Usuario				
Tipo	Primario				
Referencias	CU-03, CU-04, CU-05				
Precondición	El archivo SVG ha sido generado o editado.				
Postcondición	El archivo SVG se muestra correctamente en la interfaz del usuario.				
Autor	Alfonso Piñera Herrera	**Fecha**	21-08-24	**Versión**	1.0
Propósito	Permitir al usuario visualizar el archivo SVG generado o editado en la interfaz de la aplicación.				
Resumen	El sistema presenta el archivo SVG en la interfaz de usuario para su revisión y posible edición o guardado.				

Paso	**Descripción**
Paso 1	El sistema muestra el archivo SVG en la interfaz de usuario.
Paso 2	El usuario puede interactuar con el archivo (hacer zoom, desplazar, etc.).
Paso 3	El usuario puede decidir editar, guardar o compartir el archivo visualizado.

Cuadro 5.6: Descripción del Caso de Uso: Visualizar Archivo SVG

Caso de Uso	Guardar Configuracion JSON			**ID**	CU-07
Actor	Usuario				
Tipo	Primario				
Referencias	CU-03, CU-04				
Precondición	El archivo JSON ha sido generado y/o editado.				
Postcondición	El archivo JSON es guardado en el dispositivo del usuario.				
Autor	Alfonso Piñera Herrera	**Fecha**	21-08-24	**Versión**	1.0
Propósito	Permitir al usuario guardar el archivo JSON generado o editado en su dispositivo para su posterior uso.				
Resumen	El usuario selecciona la opción de guardar, y el sistema guarda el archivo JSON en la ubicación seleccionada por el usuario en su dispositivo.				

Paso	**Descripción**
Paso 1	El usuario selecciona la opción de guardar el archivo JSON.
Paso 2	El usuario elige la ubicación y el nombre del archivo a guardar.
Paso 3	El sistema guarda el archivo JSON en la ubicación seleccionada.

Cuadro 5.7: Descripción del Caso de Uso: Guardar Configuracion JSON

5.5. Diagramas UML

Se utilizaron diagramas UML para representar el diseño estructural y de comportamiento del sistema. Estos diagramas proporcionan una vista clara de la arquitectura y del flujo de control dentro del sistema.

5.5.1. Diagrama de Casos de Uso

El siguiente diagrama de casos de uso describe las interacciones principales entre los usuarios y el sistema:

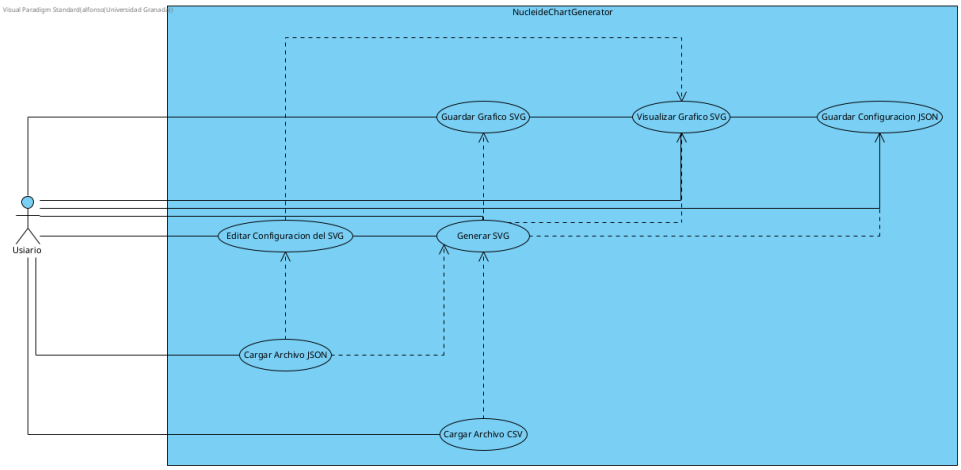


Figura 5.5: Diagrama de Casos de Uso del Sistema.

5.5.2. Diagrama de Secuencia

El diagrama de secuencia ilustra la interacción entre los objetos del sistema durante la ejecución de un proceso específico. Este diagrama es útil para comprender el flujo de control y la comunicación entre componentes.

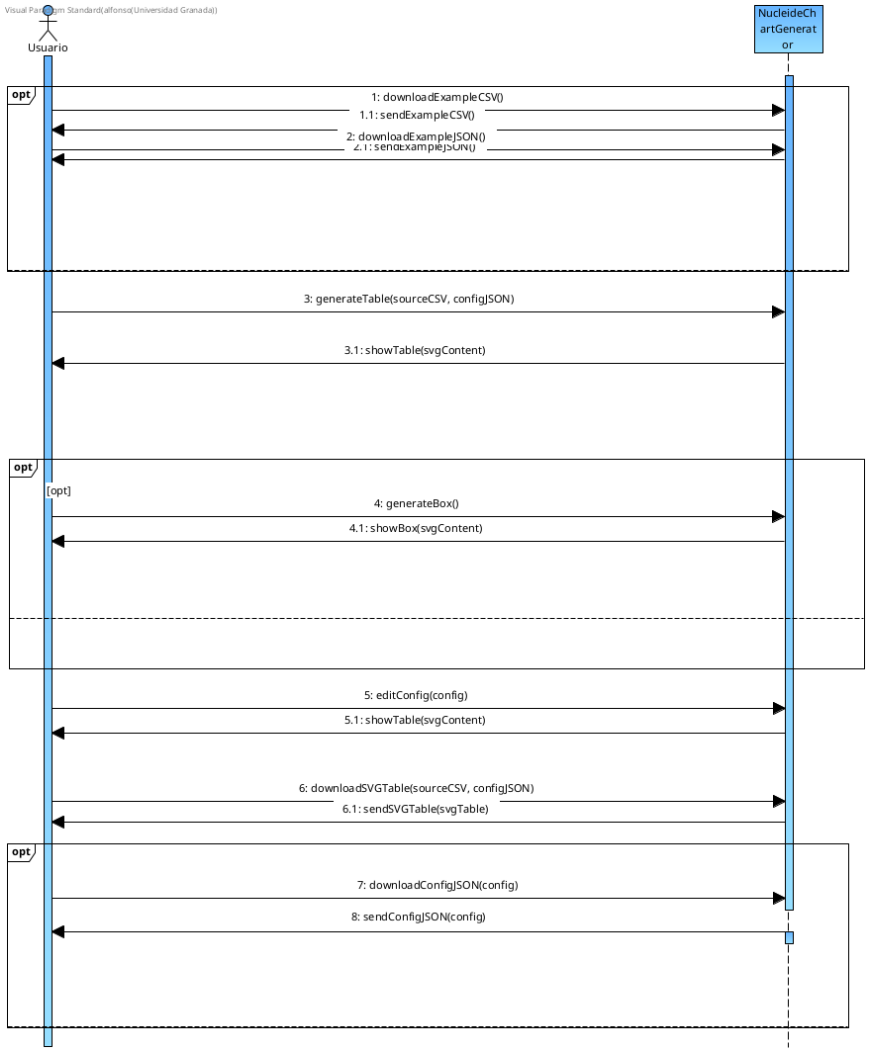


Figura 5.6: Diagrama de secuencia para un proceso clave del sistema

5.6. Consideraciones de Seguridad

Aunque el sistema no maneja datos sensibles o requiere autenticación, se implementaron medidas básicas de seguridad para garantizar la integridad y la correcta operación del sistema.

- **Validación de Entrada:** Todas las entradas del usuario son validadas para prevenir errores y asegurar la correcta operación del sistema.
- **Gestión de Errores:** Se diseñó un manejo de errores robusto para prevenir fallos inesperados y asegurar que el sistema pueda recuperarse de situaciones anómalas.

5.7. Conclusiones del Diseño

El diseño del sistema fue realizado con un enfoque en la modularidad, escalabilidad y simplicidad. Se logró un diseño que cumple con los requisitos establecidos y que puede ser mantenido y escalado fácilmente en el futuro. La implementación de principios de diseño robustos asegura que el sistema sea eficiente y fácil de usar.

Capítulo 6

Implementación

6.1. Introducción

En este capítulo se detalla el proceso de implementación del sistema, incluyendo las herramientas y tecnologías utilizadas, la estructura del código, y las decisiones técnicas clave. Se abordarán los aspectos más relevantes de la implementación, como la organización del código, los patrones de diseño aplicados y los desafíos encontrados durante el desarrollo.

6.2. Entorno de Desarrollo

El entorno de desarrollo utilizado para la implementación del sistema ha sido seleccionado para garantizar una alta productividad y calidad en el código. A continuación, se describen las herramientas y tecnologías empleadas:

- **Lenguaje de Programación:** Como fue definido en el análisis del sistema (capítulo 3), se ha hecho uso de *Python* para la parte lógica, de comunicación y de procesamiento, y *React* para la parte de la interfaz, debido a que ambas herramientas destacan por su simplicidad y la amplia disponibilidad de bibliotecas para el manejo de datos y procesamiento.
- **Editor de Código:** Se utilizó *Neovim*, un editor ligero pero potente, que ofrece soporte para múltiples lenguajes y una amplia variedad de plugins.
- **Control de Versiones:** *Git* fue utilizado para el control de versiones, permitiendo el seguimiento de cambios en el código.

- **Sistema Operativo:** El desarrollo se realizó en un entorno *Linux*, con distribuciones de Arch Linux y Debian, lo que facilitó el manejo de dependencias y el despliegue en servidores similares, tanto en el servidor de desarrollo (Debian11 self-hosted) como en el de producción (AWS EC2 Debian).

6.3. Estructura del Código

El código del sistema se organizó de manera modular, siguiendo buenas prácticas de desarrollo y asegurando la mantenibilidad y extensibilidad del mismo. A continuación se presenta la estructura de directorios del proyecto:

6.3.1. nucleidechartlib

Este directorio contiene el código fuente de la librería principal, que genera las tablas de nucleidos. Está programado en Python y hace uso de la librería *svgwrite* para generar imágenes vectoriales. La estructura de directorios de esta librería es la siguiente:

```
.
|-- data
|   |-- 2div_min.json
|   |-- 2div_regular.json
|   |-- 3div_regular.json
|   |-- description.json
|   |-- example.csv
|   |-- no_div_min.json
|   `-- no_div_regular.json
|-- nucleidechartlib
|   |-- core.py
|   |-- element
|   |   |-- element.py
|   |   |-- __init__.py
|   |   `-- nucleide.py
|   |-- enums.py
|   |-- graphics
|   |   |-- config.py
|   |   |-- element_box.py
|   |   |-- __init__.py
|   |   |-- nucleide_sect.py
|   |   `-- nucleide_table.py
|   |-- __init__.py
```

```
|   `-- requirements.txt
|-- requirements.txt
|-- setup.py
|-- tests
|   |-- __init__.py
|   |-- test_elements_boxes.py
|   |-- test_elements.py
|   `-- test_element_table.py
`-- utils
    |-- csv_driver.py
    `-- __init__.py
```

Directorio `/nucleidechartlib`

Este directorio contiene todo el código fuente de la librería que gestiona la generación de tablas de nucleidos. Está organizado en módulos, cada uno agrupando clases y funciones relacionadas, siguiendo el principio de responsabilidad única.

Directorio `/tests`

Aquí se encuentran las pruebas unitarias del sistema, implementadas utilizando la biblioteca *unittest* de Python, asegurando que cada parte del código funcione correctamente de manera aislada.

Directorio `/utils`

Este directorio contiene scripts que proporcionan utilidades adicionales a la librería, como un gestor de configuraciones y un lector de archivos CSV.

6.3.2. nucleidechartAPI

Esta sección detalla la API que utiliza la librería para responder a las peticiones del frontend generando imágenes SVG. Está programada en Python usando el microframework Flask.

6.3.3. nucleidechartfrontend

La última parte de la aplicación es el frontend, programado en React. Se encarga de mostrar la tabla de nucleidos generada por la API, permitir al usuario modificar la tabla y su configuración, y proporcionar algunos

ejemplos en formato SVG y archivos de configuración JSON, así como un archivo CSV con los datos de los nucleidos.

Módulos Implementados

Módulo de Presentación

El módulo de presentacion gestiona la interacción con servicios externos desde el frontend. A continuación se muestra un ejemplo de cómo se realiza esta comunicación en el archivo JavaScript:

```
export const generateTable = async (jsonFile, csvFile,
    csvChanged) => {
  const formData = new FormData();
  formData.append('config', jsonFile);
  if (csvChanged) {
    formData.append('source', csvFile);
  }
  formData.append('sessionID', sessionID);

  try {
    const response = await axios.post((API_URL + "gen_table/"),
        formData, {
      headers: {
        'Content-Type': 'multipart/form-data',
      },
    });
    const svg = response.data;

    const formConfigData = new FormData();
    formConfigData.append('config', jsonFile);
    formConfigData.append('sessionID', sessionID);

    const responseConfig = await axios.post((API_URL + "
        get_config/"), formConfigData, {
      headers: {
        'Content-Type': 'multipart/form-data',
      },
    });
    const config = responseConfig.data;
    console.log('API Response\nConfig:', config);
    return { svg, config };

  } catch (error) {
    console.error('Error generating SVG:', error);
    throw error;
  }
};
```

Módulo de API Este modulo esta compuesto por toda la API y es el que se encarga de responder las peticiones que se le hacen. Maneja 3 tipos de

peticiones: /gen_table/ Se encarga de generar y responder con una tabla de nucleidos al recibir un CSV y un JSON. /get_config/ Se encarga de recibir y comprobar una configuración JSON y devolver una configuracion completa. /gen_element_box/ Se encarga de generar y responder con una sola caja de un nucleido al recibir un CSV, un JSON y el elemento a generar en un string.

Módulo de Lógica de Negocio

Este módulo, ubicado en 'nucleidechartlib/core.py', se encarga del procesamiento de la información y la generación de tablas en formato SVG. Aquí se muestra un fragmento de este módulo:

```
1  from utils.csv_driver import read_elements, read_element
2  from utils.utils import update_config
3  from nucleidechartlib.graphics.nucleide_sect import
       Nucleide_Sect
4  from nucleidechartlib.graphics.nucleide_table import
       Nucleide_Table
5  from nucleidechartlib.graphics.element_box import Element_Box
6  from nucleidechartlib.graphics.config import default_config
7  import svgwrite
8  import json
9
10 def load_elements_csv(file_name):
11     return read_elements(file_name)
12
13 def load_element_csv(file_name, n):
14     element = read_element(file_name, n)
15     return element
16
17 def gen_chart(elements, output, config, description={}, style):
18     element_boxes = {}
19     id = 0
20     with open('/home/alfonso/GIT/nucleidechartlib/data/
           description.json') as json_file:
21         description = json.load(json_file)
22
23     valid_config = update_config(default_config, config)
24     for element in elements.values():
25         n = 0
26         nucleide_boxes = {}
27         for nucleide in element.nucleides.values():
28             nucleide_boxes[n] = Nucleide_Sect(id=n,
29                                               name=element.
                                                   symbol+str(
                                                   element.
                                                   mass_number)+
                                                   "-"+str(n),
30                                               half_life=
                                                   nucleide.
                                                   half_life,
31                                               decay_modes_intensities
```

```
                                                    =nucleide.
                                                    decay_modes_intensities
                                                    ,
32                                                extra={})
33              n += 1
34
35          element_boxes[id] = Element_Box(id=id,
36                                          name=element.symbol+str(
                                                element.
                                                atomic_number),
37                                          symbol=element.symbol,
38                                          mass_number=element.
                                                mass_number,
39                                          atomic_number=element.
                                                atomic_number,
40                                          nucleides=nucleide_boxes
                                                )
41          id += 1
42      table = Nucleide_Table(id=0, name="NucleideChart",
43                             cols=100, rows=70, element_boxes=
                                   element_boxes)
44
45      table.draw(filename=output, config=valid_config, style=
            style, description=description)
```

Módulo de Gestión de Datos

Este módulo se encarga de manejar la información en memoria y se encuentra en el archivo ‘utils/csv_driver.py‘. Su función principal es transformar los datos proporcionados por nucleonica en un formato adecuado. A continuación se muestra un ejemplo de este módulo:

```
1  import csv
2  import re
3  from nucleidechartlib.element.element import Element
4  from nucleidechartlib.element.nucleide import Nucleide
5  from nucleidechartlib.enums import DecayMode
6
7  def read_elements(file_name):
8      with open(file_name, newline='') as csvfile:
9          reader = csv.DictReader(csvfile)
10
11         temp_sym = ""
12         temp_Elem = Element(id=-1, symbol="dummy",
               atomic_number=-1, mass_number=-1, nucleides={})
13         id_elem = 0
14         id_nucleide = 0
15
16         elements = dict()
17         for row in reader:
18             if row['Decay Modes and their Intensities'] == '':
19                 continue
```

```
20              symbol = row['A Element']
21              if temp_sym != symbol:
22                  if temp_Elem and temp_Elem.id != -1:
23                      elements[id_elem]=temp_Elem
24                  temp_sym = symbol
25                  temp_Elem = Element(id=id_elem, symbol=
                        remove_numbers(symbol),
26                                      atomic_number=int(int(row['
                                            Atomic Number']) / 10),
27                                      mass_number=int(row['Mass
                                            Number']),
28                                      nucleides={})
29                  id_elem += 1
30
31              half_life = row['Half-life']
32              if row['Half-life unit'] != '':
33                  half_life += " " + row['Half-life unit']
34              year_of_discovery = row['Year of Discovery']
35
36              if year_of_discovery is not int():
37                  year_of_discovery = 0
38
39              mass_excess = row['Mass Excess']
40              if mass_excess is not float():
41                  mass_excess = 0.0
42
43              nucleide = Nucleide(id=id_nucleide,
44                                  mass_excess=float(mass_excess),
45                                  half_life=half_life,
46                                  decay_modes_intensities={},
47                                  energy_levels={},
48                                  spin_and_parity=row['Spin and
                                        Parity'],
49                                  year_of_discovery=int(
                                        year_of_discovery))
50              id_nucleide += 1
51
52              set_decay_modes(row['Decay Modes and their
                    Intensities'], nucleide)
53              stable = nucleide.decay_modes_intensities.get(
                    DecayMode.STABLE)
54              if stable is not None:
55                  nucleide.half_life = str(stable)
56              temp_Elem.nucleides[nucleide.id] = nucleide
57
58      elements[id_elem]=temp_Elem
59      return elements
```

6.4. Despliegue

Para el despliegue de la aplicación se han creado Docker containers para las partes de ‘nucleidechartAPI‘ y ‘nucleidechartfrontend‘. Estos Docker containers se han subido a sus respectivos repositorios y se han desplegado en un servidor AWS EC2 Debian11. Para el despliegue de la aplicación se ha hecho uso de ‘docker-compose‘, que permite desplegar los Docker containers de forma sencilla y rápida.

Ejemplo de ‘docker-compose.yml‘ para la API:

```
services:
  nucleide-chart-api:
    build:
      context: ./
      dockerfile: Dockerfile
    ports:
      - '5000:5000'
    volumes:
      - .:/usr/src/backend
```

El puerto 5000 es el puerto en el que se desplegará la API en el servidor. El volumen se utiliza para montar el directorio de la API en el servidor, facilitando el acceso a los archivos de la API y el rastreo de errores.

Ejemplo de ‘docker-compose.yml‘ para el frontend:

```
services:
  frontend:
    build:
      context: .
      dockerfile: Dockerfile
    environment:
      - API_URL=https://nucleidechart.vaelico.es/API/
    ports:
      - "3000:3000"
```

En este Docker container, el puerto de salida es el 3000, y se ha especificado la variable de entorno ‘API_URL‘ que contiene la URL de la API, permitiendo al frontend especificar la URL sin modificar el código fuente.

6.5. Desafíos y Soluciones

Durante la implementación del sistema, se encontraron varios desafíos técnicos. A continuación, se describen algunos de los más significativos y las soluciones adoptadas:

- **Gestión de Dependencias:** Mantener las dependencias del proyecto actualizadas y compatibles fue un desafío continuo. Se solucionó mediante el uso de *virtualenv* y un archivo `requirements.txt` para gestionar las versiones de las bibliotecas.
- **Optimización del Rendimiento:** Algunas operaciones de procesamiento resultaron ser más lentas de lo esperado. Se optimizaron mediante el uso de estructuras de datos adecuadas y la paralelización de tareas donde fue posible.
- **Manejo de Errores:** Garantizar que el sistema manejara adecuadamente todos los posibles errores y excepciones fue clave para la robustez del sistema. Se implementaron mecanismos de manejo de excepciones centralizados y se realizaron pruebas exhaustivas para cubrir todos los casos posibles.

6.6. Conclusiones de Implementación

La implementación del sistema fue llevada a cabo con éxito, cumpliendo con los requisitos establecidos. Se logró desarrollar un sistema modular, mantenible y escalable, capaz de cumplir con las expectativas de rendimiento y calidad.

El uso de buenas prácticas de programación, junto con una adecuada planificación y ejecución de pruebas, aseguró que el sistema fuera robusto y fiable. Se recomienda continuar con un proceso de mejora continua, revisando y refactorizando el código según sea necesario para mantener la calidad y relevancia del sistema en el tiempo.

Capítulo 7

Pruebas

7.1. Introducción

Este capítulo describe las pruebas realizadas para asegurar la calidad y el correcto funcionamiento del sistema desarrollado. Se presentan los tipos de pruebas efectuadas, los casos de prueba más representativos y los resultados obtenidos. El proceso de prueba fue esencial para identificar y corregir errores, así como para validar que el sistema cumple con los requisitos funcionales y no funcionales establecidos.

7.2. Estrategia de Pruebas

Para garantizar que el sistema cumple con las expectativas, se siguió una estrategia de pruebas bien definida que abarcó diferentes niveles de validación:

- **Pruebas Unitarias:** Aseguraron que cada componente individual del sistema funcionara correctamente en aislamiento.
- **Pruebas de Integración:** Verificaron la correcta interacción entre los diferentes módulos y componentes del sistema.
- **Pruebas de Sistema:** Evaluaron el sistema completo en condiciones que simulan el entorno de producción.
- **Pruebas de Aceptación:** Validaron que el sistema satisface los requisitos del cliente y cumple con los casos de uso definidos.

7.3. Pruebas Unitarias

Las pruebas unitarias se llevaron a cabo para verificar la funcionalidad de los componentes individuales del sistema. Se empleó la biblioteca *unittest* de Python para automatizar estas pruebas, asegurando que fueran repetibles y eficientes.

7.3.1. Cobertura de Pruebas

Se aseguró una alta cobertura de código mediante la creación de casos de prueba para todos los métodos y funciones clave del sistema. A continuación se muestra un ejemplo de prueba unitaria para la API:

```
1     def test_generar_tabla(self):
2         with io.BytesIO(b'Mass Number,Atomic Number,A Element,
              Isomer,Mass Excess,Mass Excess uncertainty,Isomer
              Excitation Energy,Isomer Excitation Energy
              uncertainty,Origin of Excitation Energy,Isom.Unc,
              Isom.Inv,Half-life,Half-life unit,Half-life
              uncertainty,Spin and Parity,Ensdf year,Year of
              Discovery,Decay Modes and their Intensities\n001
              ,0010,1H,,7288.971064,0.000013,,,,,,stbl
              ,,,1/2+*,06,1920,IS=99.9855 78\n') as test_csv:
3             response = self.client.post('/gen_table/', data={
4                 'sessionID': 'test-session-id',
5                 'source': (test_csv, 'test.csv')
6             }, content_type='multipart/form-data')
7
8             self.assertEqual(response.status_code, 200)
9             self.assertIn('Content-Disposition', response.
                  headers)
10            self.assertTrue(response.headers['Content-
                  Disposition'].startswith('attachment'))
```

7.3.2. Resultados de las Pruebas Unitarias

Las pruebas unitarias resultaron en una alta confiabilidad del código, con una gran cobertura de código y una tasa de éxito del 100 % en todos los casos de prueba definidos. Los pocos errores encontrados se debieron principalmente a casos especiales no contemplados inicialmente, los cuales fueron corregidos y reprobados satisfactoriamente.

7.4. Pruebas de Integración

Las pruebas de integración se realizaron para asegurar que los módulos del sistema interactuaran correctamente entre sí. Estas pruebas incluyeron la validación de flujos de trabajo completos y la comunicación entre diferentes partes del sistema.

7.5. Pruebas de Sistema

Las pruebas de sistema se llevaron a cabo para validar el comportamiento del sistema completo en un entorno similar al de producción. Estas pruebas incluyeron la verificación de los requisitos funcionales y no funcionales, así como pruebas de rendimiento y estabilidad.

7.5.1. Casos de Prueba de Sistema

Un ejemplo de los casos de prueba de sistema realizados:

Descripción	Evaluar la capacidad del sistema para manejar grandes volúmenes de datos sin pérdida de rendimiento.
Resultado Esperado	El sistema debe procesar todos los datos sin errores y en un tiempo aceptable.
Resultado Obtenido	Las pruebas realizadas con JMeter mostraron tiempos de respuesta menores a 2 segundos para la generación de hasta 1000 tablas de nucleidos simultaneas en el entorno de desarrollo.

Cuadro 7.1: Caso de Prueba: Procesamiento de un gran conjunto de datos

7.6. Pruebas de Aceptación

Las pruebas de aceptación fueron realizadas en colaboración con el tutor del proyecto para asegurar que el sistema cumpliera con los requisitos y expectativas definidas.

7.6.1. Proceso de Aceptación

Durante las pruebas de aceptación, se validaron todos los casos de uso descritos en las fases de análisis y diseño. El sistema fue evaluado en un

entorno controlado, verificando que todas las funcionalidades estuvieran implementadas y funcionaran correctamente.

7.6.2. Resultados de las Pruebas de Aceptación

El sistema fue aceptado por el tutor, quienes confirmaron que cumple con todos los requisitos funcionales y no funcionales especificados. Se sugirieron algunas mejoras menores, las cuales fueron incorporadas en la versión final del sistema.

7.7. Conclusiones de Pruebas

El proceso de pruebas fue fundamental para asegurar la calidad y fiabilidad del sistema. Se llevaron a cabo pruebas exhaustivas a diferentes niveles, lo que permitió identificar y corregir errores, optimizar el rendimiento y validar el cumplimiento de los requisitos.

Gracias a estas pruebas, se pudo entregar un sistema robusto y confiable, listo para ser desplegado en un entorno de producción. La alta cobertura de código y el éxito en las pruebas de aceptación garantizan que el sistema está preparado para cumplir con su propósito de manera efectiva.

Capítulo 8

Conclusiones y trabajos futuros

8.1. Logros Alcanzados

A lo largo del desarrollo del proyecto, se lograron los siguientes hitos clave:

- **Cumplimiento de Requisitos:** Se implementaron todos los requisitos funcionales y no funcionales, garantizando que el sistema cumpliera con las expectativas del cliente.
- **Integración Exitosa:** Se logró una integración fluida entre los diferentes módulos del sistema, asegurando un funcionamiento coherente y eficiente.
- **Optimización de Recursos:** Se llevaron a cabo mejoras en el uso de memoria y rendimiento, lo que permitió al sistema manejar grandes volúmenes de datos sin comprometer la velocidad o estabilidad.
- **Pruebas Exhaustivas:** El sistema fue sometido a rigurosas pruebas en todos los niveles, lo que permitió identificar y corregir errores, y asegurar un alto nivel de calidad en el producto final.
- **Aceptación del Cliente:** El sistema fue validado y aceptado por el tutor, quien confirmó que cumple con todos los requisitos y satisface las necesidades planteadas al inicio del proyecto.

RF	Resumen de Funcionalidad	Estado
RF1	Permite al usuario cargar un archivo CSV con los datos del conjunto de nucleidos.	✓
RF1	Permite al usuario cargar un archivo JSON con las configuraciones de estilo del SVG.	✓
RF2	Genera un archivo SVG basado en los datos del archivo CSV.	✓
RF2	Genera un archivo SVG basado en las configuraciones proporcionadas en el archivo JSON.	✓
RF3	Permite al usuario modificar las configuraciones del SVG en tiempo real.	✓
RF3	Los cambios en las configuraciones deben reflejarse inmediatamente en el archivo SVG generado.	✓
RF4	Permite al usuario visualizar el SVG generado en el navegador.	✓
RF4	Permite al usuario interactuar con el SVG (e.g., zoom, pan, recarga).	✓
RF5	Permite al usuario descargar el archivo SVG generado en el navegador.	✓
RF5	Permite al usuario descargar el archivo JSON con las configuraciones usadas para generar el SVG.	✓

Cuadro 8.1: Resumen de Funcionalidad de la Aplicación

8.2. Lecciones Aprendidas

Durante el desarrollo del proyecto, se aprendieron valiosas lecciones que contribuirán a futuros trabajos:

- **Importancia de la Planificación:** Una planificación detallada desde el principio es crucial para identificar riesgos y gestionar recursos de manera eficiente.
- **Flexibilidad en el Desarrollo:** Es importante mantener una actitud flexible durante el desarrollo para adaptarse a cambios en los requisitos o en el entorno de trabajo.
- **Comunicación Efectiva:** La comunicación constante y clara entre todos los miembros del equipo y los interesados es esencial para asegurar que todos los involucrados estén alineados y se minimicen malentendidos.
- **Pruebas como Pilar Fundamental:** Invertir tiempo en pruebas exhaustivas desde etapas tempranas del desarrollo permite identificar problemas de manera oportuna y reduce el costo de corrección en fases avanzadas del proyecto.

8.3. Trabajo Futuro

Aunque el sistema desarrollado cumple con los requisitos actuales, existen áreas que podrían explorarse en futuros desarrollos para mejorar y expandir las capacidades del sistema:

- **Escalabilidad:** Implementar mejoras adicionales para asegurar que el sistema pueda manejar una mayor cantidad de usuarios y datos sin perder eficiencia.

- **Seguridad:** Fortalecer las medidas de seguridad del sistema para proteger mejor los datos sensibles y garantizar la privacidad de los usuarios.
- **Experiencia de Usuario:** Continuar refinando la interfaz de usuario para hacerla más intuitiva y accesible para un público más amplio.
- **Automatización de Pruebas:** Ampliar el conjunto de pruebas automatizadas para incluir casos más complejos y mejorar la eficiencia del proceso de pruebas.
- **Soporte para Bases de Datos Adicionales:** Agregar soporte para otros motores de bases de datos para aumentar la flexibilidad y la compatibilidad con diferentes entornos.

8.4. Conclusión Final

El desarrollo de este proyecto ha sido un proceso integral que no solo ha permitido crear un sistema funcional, sino que también ha brindado valiosas experiencias y conocimientos que serán útiles en futuros proyectos. La combinación de una planificación adecuada, un enfoque iterativo en el desarrollo y una rigurosa estrategia de pruebas ha resultado en un producto final que cumple con los objetivos iniciales y proporciona una base sólida para futuras mejoras y expansiones.

Capítulo 9

Glosario

Frontend: Parte del sistema que interactúa directamente con el usuario, permitiendo la carga de archivos y la visualización de las tablas SVG generadas.

Backend: Capa de aplicación encargada de procesar los datos recibidos del frontend y de generar las tablas de nucleidos en formato SVG utilizando la lógica de negocio

API: Interfaz de programación de aplicaciones que gestiona las peticiones del frontend, procesa archivos y devuelve resultados, como la generación de tablas SVG.

SVG: Formato de gráficos vectoriales escalables utilizado para representar las tablas generadas con los datos del archivo CSV y las configuraciones del archivo JSON.

React: Biblioteca de JavaScript utilizada para desarrollar la interfaz de usuario del frontend en este proyecto. Permite la creación de componentes reutilizables.

Mockup: Representación estática de una interfaz de usuario, utilizada para planificar el diseño y la estructura de la interfaz antes de su implementación.

CI/CD: Práctica de desarrollo que incluye la integración continua (CI) y el despliegue continuo (CD), para automatizar el proceso de pruebas y despliegue en proyectos de software.

UML: Lenguaje de modelado unificado que permite representar el diseño de sistemas a través de diagramas como los de clases, secuencias y casos de uso.

Docker: Plataforma para empaquetar aplicaciones en contenedores, lo que garantiza que funcionen en cualquier entorno de manera consistente,

facilitando el despliegue.

Sprint: Periodo de tiempo en las metodologías ágiles en el que un equipo de desarrollo se enfoca en completar tareas específicas del proyecto.

JSON: Formato ligero de intercambio de datos, utilizado para almacenar y transmitir configuraciones o datos de manera estructurada en texto plano.

CSV: Formato de archivo de texto plano utilizado para representar datos tabulares, donde las columnas están separadas por comas.

Diagrama de Gantt: Herramienta de planificación que muestra visualmente las tareas de un proyecto a lo largo del tiempo, facilitando el seguimiento de plazos y dependencias.

Bibliografía

[1] Joint Research Centre. (n.d.). Karlsruhe Nuclide Chart. Recuperado de https://joint-research-centre.ec.europa.eu/tools-and-laboratories/training-programmes/karlsruhe-nuclide-chart_en

[2] International Atomic Energy Agency. (n.d.). Nuclide Chart Viewer. Recuperado de https://www-nds.iaea.org/relnsd/vcharthtml/VChartHTML.html

[3] International Atomic Energy Agency. (n.d.). LiveChart of Nuclides: Advanced Version. Recuperado de https://www.iaea.org/resources/databases/livechart-of-nuclides-advanced-version

[4] Wikipedia. (n.d.). UTF-8. Recuperado de https://en.wikipedia.org/wiki/UTF-8.

[5] Junta de Andalucía. (2024, 25 de febrero). Guía para la redacción de casos de uso. Recuperado de https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416.

[6] es.talent.com. (2024). Salario medio por trabajo en España, 2024. Recuperado de https://es.talent.com/salary?job=2024.

[7] BOE. (2014). Ley 27/2014, de 27 de noviembre, del Impuesto sobre Sociedades. Recuperado de https://www.boe.es/buscar/act.php?id=BOE-A-2014-12328.

[8] Australian National University. (n.d.). Nuclide Chart. Recuperado de https://people.physics.anu.edu.au/~ecs103/chart/.

[9] React Documentation. (n.d.). Learn React. Recuperado de https://es.react.dev/learn.

[10] Midudev. (n.d.). Aprendiendo React. Recuperado de https://github.com/midudev/aprendiendo-react.

[11] Docker Documentation. (n.d.). Orientation and setup. Recuperado de https://docker-docs.uclv.cu/get-started/.

[12] Node.js Documentation. (n.d.). Documentation. Recuperado de https://nodejs.org/en/docs/.

[13] Fondo de la portada. Recuperado de https://www.freepik.es/vector-gratis/vector-fondo-tecnologia-digital-marco-hexagonal-tono-azul_16252198.htm#from_view=detail_alsolike.

Nucleide Chart Generator

Generated by: nucleidechart.vaelico.es

Version: 1.0

Author: Alfonso Jesús Piñera Herrera

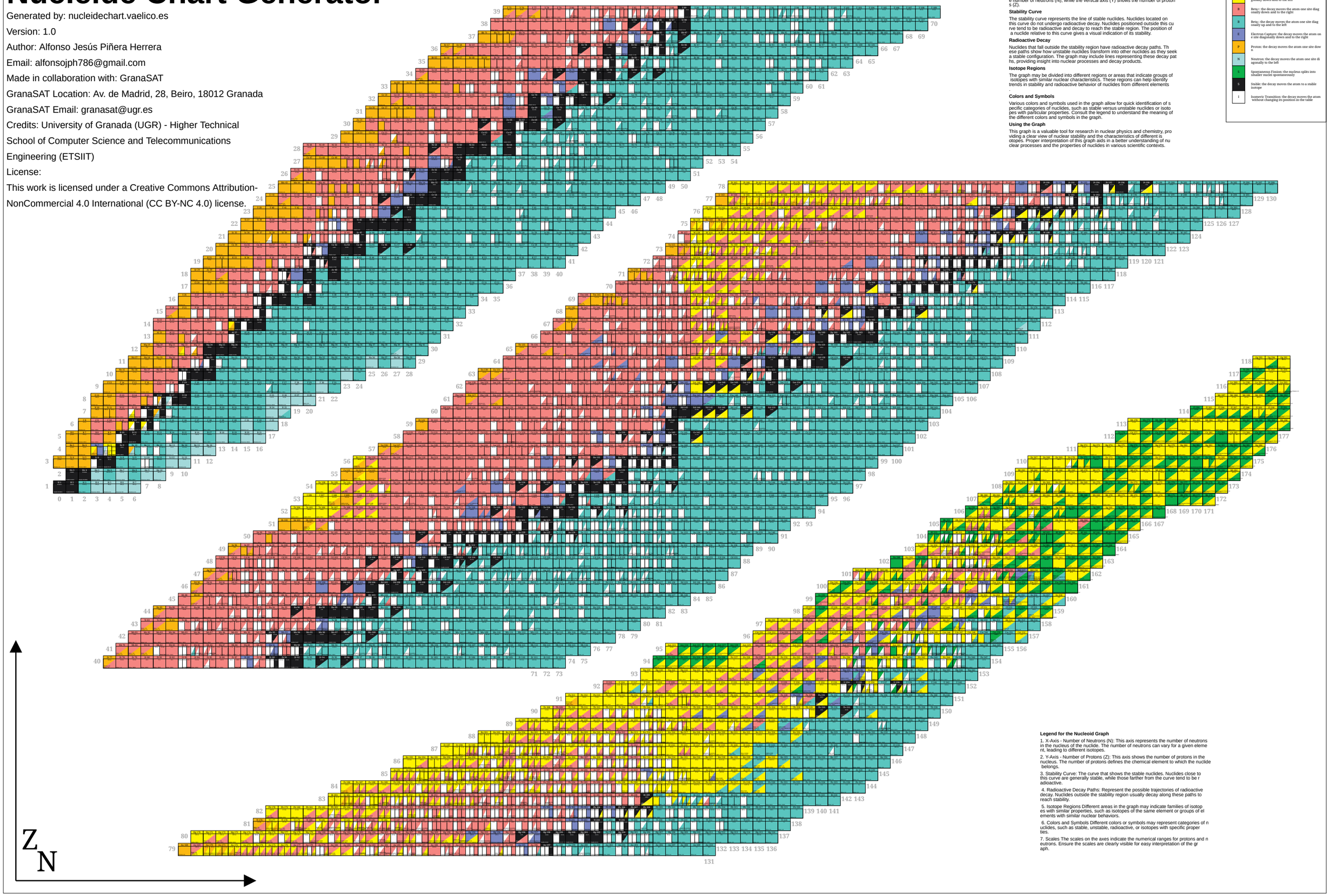
Email: alfonsojph786@gmail.com

Made in collaboration with: GranaSAT

GranaSAT Location: Av. de Madrid, 28, Beiro, 18012 Granada

GranaSAT Email: granasat@ugr.es

Credits: University of Granada (UGR) - Higher Technical School of Computer Science and Telecommunications Engineering (ETSIIT)



General Description

The nucleoid graph provides a visual representation of nuclides (isotopes) based on their numbers of protons and neutrons. The horizontal axis (X) represents the number of neutrons (N), while the vertical axis (Y) shows the number of protons (Z).

Stability Curve

The stability curve represents the line of stable nuclides. Nuclides located on this curve do not undergo radioactive decay. Nuclides positioned outside this curve tend to be radioactive and decay to reach the stable region. The position of a nuclide relative to this curve gives a visual indication of its stability.

Radioactive Decay

Nuclides that fall outside the stability region have radioactive decay paths. These paths show how unstable nuclides transform into other nuclides as they seek a stable configuration. The graph may include lines representing these decay paths, providing insight into nuclear processes and decay products.

Isotope Regions

The graph may be divided into different regions or areas that indicate groups of isotopes with similar nuclear characteristics. These regions can help identify trends in stability and radioactive behavior of nuclides from different elements

Colors and Symbols

Various colors and symbols used in the graph allow for quick identification of specific categories of nuclides, such as stable versus unstable nuclides or isotopes with particular properties. Consult the legend to understand the meaning of the different colors and symbols in the graph.

Using the Graph

This graph is a valuable tool for research in nuclear physics and chemistry, providing a clear view of nuclear stability and the characteristics of different isotopes. Proper interpretation of this graph aids in a better understanding of nuclear processes and the properties of nuclides in various scientific contexts.

Legend

- A: Alpha: the decay moves the atom two sites diagonally down and to the left
- B-: Beta-: the decay moves the atom one site diagonally down and to the right
- B+: Beta+: the decay moves the atom one site diagonally up and to the left
- E: Electron Capture: the decay moves the atom one site diagonally down and to the right
- P: Proton: the decay moves the atom one site down
- N: Neutron: the decay moves the atom one site diagonally to the left
- S: Spontaneous Fission: the nucleus splits into smaller nuclei spontaneously
- S: Stable: the decay moves the atom to a stable isotope
- I: Isomeric Transition: the decay moves the atom without changing its position in the table

Legend for the Nucleoid Graph

1. X-Axis - Number of Neutrons (N): This axis represents the number of neutrons in the nucleus of the nuclide. The number of neutrons can vary for a given element, leading to different isotopes.

2. Y-Axis - Number of Protons (Z): This axis shows the number of protons in the nucleus. The number of protons defines the chemical element to which the nuclide belongs.

3. Stability Curve: The curve that shows the stable nuclides. Nuclides close to this curve are generally stable, while those farther from the curve tend to be radioactive.

4. Radioactive Decay Paths: Represent the possible trajectories of radioactive decay. Nuclides outside the stability region usually decay along these paths to reach stability.

5. Isotope Regions Different areas in the graph may indicate families of isotopes with similar properties, such as isotopes of the same element or groups of elements with similar nuclear behaviors.

6. Colors and Symbols Different colors or symbols may represent categories of nuclides, such as stable, unstable, radioactive, or isotopes with specific properties.

7. Scales The scales on the axes indicate the numerical ranges for protons and neutrons. Ensure the scales are clearly visible for easy interpretation of the graph.