



UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
DIRECCIÓN DE INFORMATIZACIÓN, FACULTAD DE TECNOLOGÍAS INTERACTIVAS

SISTEMA DE GESTIÓN PARA EL INGRESO Y DISTRIBUCIÓN DE LOS NUEVOS ESTUDIANTES DE LA UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS (UCI)

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Nelson Castroman Aguiar

Gabriela de la Caridad Rodríguez Rey

Tutores: M. Sc. Ángel Alberto Vázquez Sánchez

Ing. Jorge Arias Sojo

La Habana, 2024

*La innovación es la capacidad de ver el cambio como una oportunidad, no
una amenaza. Steve Jobs*

Dedicatoria

A mi madre que me enseñó levantarme después de cada caída y a mi pareja que me ha ayudado a hacerlo.

Agradecimientos

A mi nuestros por estar siempre ahí conmigo dándolo todo, a nuestros compañeros de la uci, nuestra otra familia, sin ustedes no hubiera sido posible: Alba, Damián, Ibrahim, Yorgendy, Orlián, Susel y Nathan que ya no están con nosotros aquí pero nos animaron a lo largo del camino, a nuestros amigos que siguen presentes en nuestras vidas, a mi papá, a mi tío y abuela. A mi amiga Lelieth. A mi pareja, que nos hemos apoyado hasta por fin llegar a este momento. Gracias a todos.

Declaración de autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

Para que así conste firmamos la presente a los 2 días del mes de diciembre del año 2024.



Nelson Castroman Aguiar

Autor

Firmado por Angel Alberto Vazquez
Sánchez el día 28/11/2024 con un
certificado emitido por ROOT AC UCI

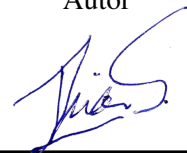
M. Sc. Ángel Alberto Vázquez Sánchez

Tutor



Gabriela de la Cajidad Rodríguez Rey

Autor



Ing. Jorge Arias Sojo

Tutor

La optimización de los procesos administrativos en los centros educativos es crucial para garantizar un acceso equitativo y mejorar la transparencia. En este contexto, los sistemas de gestión de ingreso y distribución de estudiantes juegan un papel vital, permitiendo decisiones basadas en datos y adaptaciones rápidas a cambios, como los derivados de la pandemia. El presente trabajo tiene como objetivo el desarrollo de un sistema de gestión para controlar el proceso de ingreso y realizar la distribución equitativa de los estudiantes que ingresan a la Universidad de las Ciencias Informáticas (UCI). Para ello, se seleccionan las herramientas de desarrollo Django y React.js, debido a su robustez y flexibilidad. La metodología utilizada fue Extreme Programming (XP), permitiendo obtener una aplicación con las funcionalidades requeridas por el cliente, optimizando el proceso administrativo y asegurando la continuidad y calidad del aprendizaje.

Palabras clave: Gestión, sistema, ingreso, distribución, algoritmo genético.

Introducción	1
1 Fundamentación Teórica	4
1.1 Conceptos fundamentales	4
1.2 Análisis de sistemas homólogos	5
1.3 Herramientas y tecnologías	6
1.3.1 Metodología de Desarrollo utilizada	7
1.3.2 Herramientas Case	11
1.3.3 Lenguajes de programación	11
1.3.4 Marcos de Trabajo	15
1.3.5 Gestión de Base de Datos	20
1.3.6 Editor de código	22
2 Análisis y diseño de la solución	23
2.1 Descripción del sistema a desarrollar	23
2.2 Relación de usuarios con el sistema	24
2.3 Fase de exploración	25
2.3.1 Requerimientos no funcionales	25
2.3.2 Historias de usuario	26
2.4 Fase de planificación	27
2.4.1 Estimación de esfuerzo por Historia de Usuario	28
2.4.2 Plan de iteraciones	28
2.4.3 Plan de duración de las iteraciones	28
2.4.4 Plan de entregas	28
2.5 Fase de diseño	29
2.5.1 Patrones de arquitectura de software	29
2.5.2 Patrones de diseño	31
2.5.3 Tarjetas CRC	36
2.6 Algoritmo de Inteligencia Artificial	37
2.6.1 Algoritmo Genético	39

3 Implementación y pruebas	42
3.1 Fase de Implementación	42
3.1.1 Tareas ingenieriles.	42
3.2 Fase de Pruebas	43
3.2.1 Pruebas de aceptación	44
3.2.2 Pruebas unitarias	46
Conclusiones	50
Recomendaciones	51
Referencias bibliográficas	52
Apéndices	55
A Historias de Usuario	56
B Tarjetas CRC	61
C Tareas de Ingeniería	64
D Pruebas Unitarias	68
E Pruebas de Aceptación	80
F Indicadores	93

Índice de figuras

2.1	Mapa de Navegación	24
2.2	Patrón de Arquitectura Cliente-Servidor	31
2.3	Ejemplo de bajo acoplamiento entre clases	34
2.4	Ejemplo patrones GoF	35
3.1	Pruebas de Aceptación	46
3.2	Pruebas unitarias realizadas a la ruta usuario	47
3.3	Pruebas unitarias realizadas a la ruta persona	48
3.4	Pruebas Unitarias	49
D.1	Resultado AreadeconocimientoTests	68
D.2	Resultado CentroTests	69
D.3	Resultado DoctorTests	70
D.4	Resultado DoctorandoTests	71
D.5	Resultado GraduadoTests	72
D.6	Resultado PaisTests	73
D.7	Resultado PersonaTests	74
D.8	Resultado ProgramaTests	75
D.9	Resultado SectorestTests	76
D.10	Resultado TutorTests	77
D.11	Resultado SesionTests	78
D.12	Resultado UsuarioTests	79
F.1	Entrevista realizada a subdirector de la Dirección de Informatización de la Universidad de las Ciencias Informáticas (UCI)	93

Índice de tablas

1.1	Comparación entre sistemas homólogos	6
1.2	Comparación entre tipos de metodologías existentes (Fuente: Pressman & Maxim, 2020) . . .	8
1.3	Comparación entre tipos de metodologías existentes	10
1.4	Comparación entre los lenguajes para apartado del Backend	13
1.5	Comparación de marcos de trabajo de apartado del backend	17
1.6	Comparación de Marcos de trabajo del apartado del frontend	19
1.7	Comparación de Gestores de bases de datos	20
2.1	Relación de usuarios con el sistema	24
2.2	Historia de usuario # 1	26
2.3	Historia de usuario # 2	27
2.4	Historia de usuario # 3	27
2.5	Plan de lanzamiento	28
2.6	Plan de entrega	29
2.7	Tarjeta CRC # 1	36
2.8	Tarjeta CRC # 2	36
2.9	Tarjeta CRC # 3	37
2.10	Comparación entre los algoritmos de inteligencia artificial	38
3.1	Tarea de ingeniería # 1	43
3.2	Tarea de ingeniería # 2	43
3.3	Tarea de ingeniería # 3	43
3.4	Prueba de aceptación # 1	44
3.5	Prueba de aceptación # 2	44
3.6	Prueba de aceptación # 3	45
3.7	Prueba de aceptación # 4	45
A.1	Historia de usuario # 4	56
A.2	Historia de usuario # 5	56
A.3	Historia de usuario # 6	56
A.4	Historia de usuario # 7	57

A.5	Historia de usuario # 8	57
A.6	Historia de usuario # 9	57
A.7	Historia de usuario # 10	58
A.8	Historia de usuario # 11	58
A.9	Historia de usuario # 12	58
A.10	Historia de usuario # 13	59
A.11	Historia de usuario # 14	59
A.12	Historia de usuario # 15	59
B.1	Tarjeta CRC # 4	61
B.2	Tarjeta CRC # 5	61
B.3	Tarjeta CRC # 6	61
B.4	Tarjeta CRC # 7	62
B.5	Tarjeta CRC # 8	62
B.6	Tarjeta CRC # 9	62
C.1	Tarea de ingeniería # 4	64
C.2	Tarea de ingeniería # 5	64
C.3	Tarea de ingeniería # 6	64
C.4	Tarea de ingeniería # 7	65
C.5	Tarea de ingeniería # 8	65
C.6	Tarea de ingeniería # 9	65
C.7	Tarea de ingeniería # 10	65
C.8	Tarea de ingeniería # 11	66
C.9	Tarea de ingeniería # 12	66
C.10	Tarea de ingeniería # 13	66
C.11	Tarea de ingeniería # 14	66
C.12	Tarea de ingeniería # 15	67
E.1	Prueba de aceptación # 5	80
E.2	Prueba de aceptación # 6	80
E.3	Prueba de aceptación # 7	81
E.4	Prueba de aceptación # 8	81
E.5	Prueba de aceptación # 9	81
E.6	Prueba de aceptación # 10	82
E.7	Prueba de aceptación # 11	82
E.8	Prueba de aceptación # 12	82
E.9	Prueba de aceptación # 13	83
E.10	Prueba de aceptación # 14	83

E.11 Prueba de aceptación # 15	83
E.12 Prueba de aceptación # 16	84
E.13 Prueba de aceptación # 17	84
E.14 Prueba de aceptación # 18	84
E.15 Prueba de aceptación # 19	85
E.16 Prueba de aceptación # 20	85
E.17 Prueba de aceptación # 21	85
E.18 Prueba de aceptación # 22	86
E.19 Prueba de aceptación # 23	86
E.20 Prueba de aceptación # 24	86
E.21 Prueba de aceptación # 25	87
E.22 Prueba de aceptación # 26	87
E.23 Prueba de aceptación # 27	88
E.24 Prueba de aceptación # 28	88
E.25 Prueba de aceptación # 29	88
E.26 Prueba de aceptación # 30	89
E.27 Prueba de aceptación # 31	89
E.28 Prueba de aceptación # 32	89
E.29 Prueba de aceptación # 33	90
E.30 Prueba de aceptación # 34	90
E.31 Prueba de aceptación # 35	90
E.32 Prueba de aceptación # 36	91
E.33 Prueba de aceptación # 37	91
E.34 Prueba de aceptación # 38	91
E.35 Prueba de aceptación # 39	92
E.36 Prueba de aceptación # 40	92

En la actual era digital, las Tecnologías de la Información y la Comunicación (TIC) se han convertido en el alma de la transformación educativa, marcando una diferencia sustancial en la manera en que se gestionan las instituciones académicas. Ya no es suficiente con seguir los métodos tradicionales; la evolución exige que las escuelas, colegios y universidades se adapten a nuevas formas de administrar y optimizar sus actividades diarias. Aquí es donde la gestión informática se presenta como una herramienta indispensable, permitiendo que los procesos administrativos no solo sean más eficientes, sino también más efectivos y personalizados. Los sistemas donde las tareas administrativas, como la inscripción y distribución de estudiantes y la generación de informes se realizan con una precisión quirúrgica, minimizando errores y reduciendo la carga de trabajo manual. Esta automatización no solo asegura que las operaciones se ejecuten sin contratiempos, sino que también libera un valioso tiempo para el personal.

La gestión informática en la educación va más allá de simplemente hacer las cosas más rápido o con menos esfuerzo; se trata de construir un entorno donde todos los actores involucrados, desde los estudiantes hasta los administradores, puedan tomar decisiones más informadas y basadas en datos precisos. La capacidad de analizar en tiempo real el rendimiento académico, identificar patrones y ajustar estrategias educativas es una ventaja que solo las herramientas digitales pueden ofrecer.

En la actualidad existen numerosos sistemas que se encargan de realizar esas gestiones, van desde el proceso de automatizar la matrícula de los estudiantes y la administración de sus horarios hasta el seguimiento de sus calificaciones y la impartición de sus clases. En Cuba hay sistemas que con algunas similitudes se encargan de cubrir parcialmente esas cuestiones, como el Entorno Virtual de Aprendizaje de la Universidad de las Ciencias Informáticas (UCI) donde los estudiantes acceden a sus cursos y a materiales de sus asignaturas o Akademos donde se encuentra toda su información y calificaciones, no solo de estudiantes sino también de profesores.

El actual Sistema de Ingreso de la Universidad de las Ciencias Informáticas (UCI) fue concebido como una herramienta para automatizar la matrícula de estudiantes de nuevo ingreso, su distribución en las distintas facultades y grupos, y la generación de reportes específicos relacionados con estas actividades. No obstante, el sistema presenta serias deficiencias que comprometen su utilidad y efectividad, convirtiéndolo en un obstáculo más que en un aliado para los usuarios responsables de estos procesos.

En primer lugar, la tecnología utilizada en su desarrollo se encuentra obsoleta, lo que dificulta su mantenimiento, integración con sistemas modernos y la incorporación de nuevas funcionalidades. Además, el sistema carece de documentación técnica, lo que representa un problema significativo para su análisis, modi-

ficación o mejora. Esta carencia impide que los desarrolladores comprendan su estructura interna y dificulta la detección y solución de errores.

Otro punto crítico es que el sistema no ha recibido mantenimiento ni actualizaciones durante más de cinco años, lo que ha llevado a la pérdida parcial o total de funcionalidades clave. Por ejemplo, actualmente no permite realizar correctamente la distribución de estudiantes por tipo de curso ni generar reportes detallados, procesos fundamentales en la gestión académica. Asimismo, carece de buscadores funcionales en sus interfaces, obligando a los usuarios a realizar búsquedas de manera manual, incrementando la posibilidad de errores humanos.

Más preocupante aún, algunas funcionalidades que están operativas presentan un diseño ineficiente. Un ejemplo ilustrativo es la creación de grupos: el sistema asigna nombres al azar a los grupos, y como no existe una opción para editarlos, el usuario debe modificar estos nombres manualmente accediendo directamente a la base de datos, tarea que consume tiempo y requiere conocimientos técnicos adicionales.

Estas deficiencias no solo generan frustración en los usuarios, sino que también ponen en riesgo la integridad y precisión de datos sensibles, como los registros de los estudiantes. La falta de automatización en procesos clave contradice el propósito original del sistema como herramienta de gestión, ocasionando pérdidas de tiempo, disminución de la productividad y aumentando la probabilidad de errores en la toma de decisiones. En el análisis de todo lo antes expuesto se identifica como **problema de investigación**: ¿Cómo facilitar la gestión del ingreso y distribución de estudiantes a la UCI para mejorar la eficiencia y precisión en el proceso de matrícula? Teniendo como **objeto de estudio**: El proceso de ingreso y distribución de estudiantes para instituciones educativas donde el **campo de acción** se ve reflejado en el proceso de gestión de ingreso y distribución de estudiantes en la UCI. Para darle solución al problema se define como **objetivo general**: El desarrollo de un Sistema de Gestión para el ingreso y distribución de los nuevos estudiantes de la Universidad de las Ciencias Informáticas (UCI). Para el cumplimiento de los objetivos se plantearon los siguientes **objetivos específicos**:

- Determinación de los conceptos asociados al dominio del problema de la investigación, Identificación de aplicaciones informáticas similares existentes en el ámbito nacional e internacional y selección de tecnologías y herramientas a utilizar para el proceso de desarrollo del software y determinar funcionalidades estándar que debe presentar el proyecto.
- Caracterización del proceso de gestión del ingreso y distribución de estudiantes para modelar el sistema a desarrollar.
- Desarrollo de la propuesta de solución para dar respuesta a la problemática existente.
- Validación de la solución mediante la realización de pruebas de software para comprobar la efectividad del desarrollo realizado y el cumplimiento con los objetivos propuestos.

Los resultados esperados son:

- El Sistema para la gestión del ingreso y distribución de los nuevos estudiantes de la Universidad de las Ciencias Informáticas (UCI)

- El documento de investigación que avale la realización del sistema.
- Los artefactos generados durante el desarrollo de la investigación.

Los **métodos científicos** utilizados durante la investigación fueron:

El **analítico-sintético** se emplea para determinar conceptos asociados a la problemática planteada, así como en el estudio de las soluciones existentes y la selección de las herramientas y técnicas permitiendo establecer las relaciones que existen entre los elementos de cada contexto, además en el análisis y síntesis de la documentación utilizada para la bibliografía.

Observación: se emplea para identificar algunas características en el proceso de gestión, así como las técnicas, herramientas y actividades empleadas y personas que intervienen.

Entrevista: Para el desarrollo de este método se entrevistó a Ing. Jorge Arias Sojo quien ha aportado elementos investigativos desde su experiencia como usuario del antiguo sistema.

El presente trabajo está compuesto por tres capítulos, estructurados de la siguiente forma:

- **Capítulo 1 Fundamentación teórica:** Referente a la Fundamentación Teórica. En él se describen los términos y conceptos importantes asociados al proyecto, así como elementos esenciales de las aplicaciones existentes para realizar dicho proceso y las herramientas a utilizar en el proyecto.
- **Capítulo 2 Elaboración de la propuesta de solución:** En este capítulo se comienza con la fase de planificación, donde se definen los requisitos a través de historias de usuario, priorizando las necesidades del cliente. A continuación, se describe la arquitectura del sistema y se detallan los patrones de diseño empleados, incluyendo el uso de tarjetas CRC (Clase, Responsabilidad, Colaborador) para definir las responsabilidades y colaboraciones de las clases.
- **Capítulo 3 Validación de la propuesta:** En este capítulo se detalla el proceso de validación del sistema desarrollado. Se describen las pruebas realizadas para asegurar que el código cumple con las especificaciones definidas por el cliente y se evalúa la calidad del software a través de la verificación del cumplimiento de los requisitos. Además, se analizan los resultados obtenidos y se discuten las iteraciones necesarias para garantizar el perfecto funcionamiento del sistema.

Fundamentación Teórica

El presente capítulo se exponen los conceptos fundamentales asociados al dominio del problema planteado, con el objetivo de analizar en el marco teórico el tema del objeto de estudio. Se realiza el análisis del estado del arte de los diferentes sistemas homólogos encontrados a nivel nacional e internacional, conociendo sus ventajas, desventajas y las características más importantes de estos sistemas que sean de interés para la solución de nuestro sistema. Se abordan, además, las principales características de las tecnologías y herramientas que serán utilizadas para la implementación de la solución propuesta.

1.1. Conceptos fundamentales

Con el objetivo de realizar un mejor entendimiento de los elementos relacionados a los procesos de gestión de servicios y gestión administrativa se abordarán a continuación una serie de conceptos básicos relacionados con el tema:

Se define como **sistema gestión de servicios** al conjunto de políticas, procesos y procedimientos utilizados para establecer, implementar, mantener y mejorar la gestión de servicios de TI (Tecnologías de la información) de alta calidad (Landri, 2020).

Se define como **gestión administrativa** a la organización y coordinación de los recursos y procesos dentro de las instituciones educativas. La ANT (*Actor-Network Theory*) analiza cómo las prácticas administrativas se construyen a través de redes de actores y tecnologías que influyen en la toma de decisiones, la implementación de políticas y la administración de recursos (ibíd.).

La ANT considera que la **gestión administrativa** no solo involucra a las personas (como directores y administradores), sino también a las tecnologías, documentos y procesos que forman parte del sistema administrativo. Se enfoca en cómo estas redes de actores y materiales contribuyen a la configuración y funcionamiento de la gestión administrativa en la educación (Landri, 2020).

Con el objetivo de realizar un mejor entendimiento de los procesos relacionados descritos en la aplicación a desarrollar a continuación se muestran una serie de conceptos relacionados al negocio usados para el proceso implementado en la solución:

En el contexto educativo, la **matrícula** se refiere al proceso mediante el cual los estudiantes se inscriben en una institución educativa. Incluye la inscripción formal y el registro en cursos y programas académicos (ibíd.).

El **ingreso**, en el contexto de la educación, se refiere al proceso mediante el cual los estudiantes son aceptados y comienzan a formar parte de una institución educativa (ibíd.).

La **matrícula** es una fase del proceso de ingreso. Mientras que el ingreso cubre la aceptación y registro de nuevos estudiantes, la matrícula se enfoca en la inscripción específica de los estudiantes en los cursos o programas. En esencia, el ingreso es el proceso general de aceptar a los estudiantes en la institución, mientras que la matrícula es el paso en el que los estudiantes formalizan su participación en cursos específicos (ibíd.).

1.2. Análisis de sistemas homólogos

Mediante la realización de un análisis de características presentes en el proyecto a desarrollar se identifican una serie de softwares que comparten funcionalidades y objetivos e incluso metodología de gestión de servicios principalmente en el proceso de gestión de matrícula y distribución de estudiantes en centros educativos a nivel internacional, a continuación, se muestra algunos de estos:

- **PowerSchool:** Un sistema de gestión educativa ampliamente utilizado en escuelas K-12 . Ofrece herramientas para el registro de matrículas, la gestión de clases, el seguimiento del rendimiento de los estudiantes y la comunicación con los padres. El sistema proporciona gestión de matrículas, horarios, informes de progreso, y un portal para padres y estudiantes (*PowerSchool K-12 Software & Cloud-Based Solutions 2024*).
- **EduAdmin:** Un sistema de gestión de matrículas y estudiantes utilizado principalmente en centros de formación y academias. Facilita la inscripción en cursos, la asignación de clases y la gestión de pagos. El sistema proporciona inscripción en línea, gestión de clases, control de asistencia, y procesamiento de pagos (*Acerca de EduAdmin 2024*).
- **Jenzabar:** Es una suite de software desarrollada para la gestión de instituciones educativas, especialmente orientada a universidades y colegios. Su objetivo es proporcionar una solución integral para manejar diversos aspectos administrativos y académicos en el entorno educativo (*Higher Education Software Solutions - Jenzabar 2024*).

En Cuba existe un software funcional en este ámbito. El Sistema Ingreso el cual es un sistema de la Universidad de las Ciencias Informáticas (UCI) que se encarga de gestionar el proceso de ingreso y matrícula de los estudiantes cada nuevo curso escolar y su distribución en cuanto a tipo de curso escolar, carrera, facultad y grupo. Esta aplicación ha sido explotada durante muchos años, actualmente la tecnología con la que fue desarrollada se encuentra obsoleta y aunque el sistema se encuentra funcional, varias de sus funcionalidades presentan fallos.

A continuación, se muestra una tabla que evidencia una comparación del funcionamiento de los sistemas mencionados anteriormente:

Tabla 1.1. Comparación entre sistemas homólogos

Criterio	PowerSchool	EduAdmin	Jenzabar	Sistema Ingreso
Escalabilidad	Alta	Moderada	Alta	Baja
Acceso Gratuito	Parcial	No disponible	No disponible	Disponible
Seguridad	Alta	Moderada	Alta	Moderada
Funcionalidades	Amplias (gestión académica, administrativa y financiera)	Básicas (énfoque en gestión académica)	Amplias (gestión académica, administrativa, financiera, y recursos humanos)	Amplias (gestión académica y administrativa)
Eficiencia	Alta	Moderada	Alta	Baja
Precisión	Alta	Moderada	Alta	Baja

El análisis comparativo revela varias áreas de oportunidad y mejores prácticas que pueden ser adoptadas en el sistema de gestión para el ingreso y distribución de los nuevos estudiantes:

- **Funcionalidades Esenciales:** El sistema debe permitir la inscripción de estudiantes de manera sencilla y eficiente, gestionando el proceso de matrícula completo. Esto incluye la capacidad de manejar la documentación necesaria y registrar la información de los estudiantes.
- **Automatización de Procesos:** El sistema debe automatizar tareas repetitivas y debería incluir herramientas para la generación automática de informes y análisis de datos relacionados con la inscripción y distribución de estudiantes.
- **Integración con otros Sistemas:** Debe tener capacidad para integrarse con otros sistemas de la universidad como Akademos.
- **Accesibilidad y Experiencia del Usuario:** La interfaz debe ser fácil de usar para el personal administrativo, minimizando la curva de aprendizaje y mejorando la eficiencia en la administración del sistema.
- **Seguridad de la Información:** El sistema debe implementar medidas robustas para proteger la información. Esto incluye encriptación de datos, autenticación segura y acceso controlado.
- **Soporte Técnico:** Proporcionar un soporte técnico continuo para resolver problemas y mantener el sistema actualizado.

1.3. Herramientas y tecnologías

Basado en el análisis de las características del proyecto en cuestión, y considerando diversas tendencias internacionales en el desarrollo de software, se tomaron decisiones respecto a las herramientas a emplear en el desarrollo del sistema. Estas decisiones fueron guiadas por la participación activa del cliente en cada una de las fases del proyecto, con el objetivo de implementar la solución de manera rápida y eficiente.

A continuación, se presentarán los datos relacionados con la selección de la metodología, tecnologías y herramientas utilizadas para el desarrollo del sistema de gestión para el ingreso y distribución de los nuevos estudiantes de la UCI.

1.3.1. Metodología de Desarrollo utilizada

Las metodologías de desarrollo de software se dividen principalmente en dos categorías: ágiles y tradicionales. Ambos enfoques proporcionan marcos estructurados para gestionar y ejecutar proyectos, pero difieren significativamente en sus principios y aplicaciones (Pressman y Maxim, 2020).

Por un lado, las metodologías ágiles se caracterizan por ser iterativas e incrementales, lo que permite una gran adaptabilidad y flexibilidad en proyectos donde los requisitos pueden cambiar con frecuencia. Este enfoque se enfoca en la colaboración constante entre el equipo de desarrollo y los clientes, facilitando la entrega temprana y continua de software funcional. Ejemplos destacados de metodologías ágiles incluyen Scrum y XP, que promueven la interacción frecuente y la capacidad de respuesta ante cambios en el proyecto (ibíd.).

Por otro lado, las metodologías tradicionales, como RUP, siguen un enfoque lineal y secuencial. Estas metodologías son más adecuadas para proyectos con requisitos bien definidos desde el inicio, donde la estabilidad y la previsibilidad son prioritarias. En estos casos, el desarrollo del software avanza a través de fases claramente delimitadas, cada una de las cuales debe completarse antes de pasar a la siguiente etapa. Este enfoque es menos flexible en cuanto a cambios imprevistos, pero puede ser ideal para proyectos donde es crucial una planificación detallada y proactiva (Pressman y Maxim, 2020).

Teniendo en cuenta estas diferencias fundamentales, se procedió a realizar una comparación de las características de ambos enfoques con el objetivo de determinar cuál se adaptaba mejor a las necesidades específicas del desarrollo de este proyecto:

Tabla 1.2. Comparación entre tipos de metodologías existentes (Fuente: Pressman & Maxim, 2020)

Aspectos	Ágiles	Tradicionales
Enfoque	Las metodologías ágiles se enfocan en un enfoque incremental e iterativo, centrado en la flexibilidad y adaptación continua a los cambios.	Las metodologías tradicionales se enfocan en un enfoque lineal y secuencial, centrado en seguir un plan predefinido y una estructura rígida.
Planificación	Las metodologías ágiles se enfocan en una planificación adaptativa y continua, las iteraciones cortas permiten ajustes frecuentes basados en la retroalimentación.	Las metodologías tradicionales se enfocan en una planificación exhaustiva al inicio del proyecto, los cambios posteriores son difíciles y costosos.
Análisis de requerimientos	Los requerimientos se desarrollan de manera evolutiva; se ajustan a medida que el proyecto avanza y se aprende más.	Las metodologías tradicionales se enfocan en un análisis de requerimientos detallados definidos al principio, lo que significa poca flexibilidad para cambios durante el desarrollo.
Pruebas	Las metodologías ágiles se enfocan en pruebas continuas y automatizadas durante todo el ciclo de desarrollo, lo que significa integración y pruebas frecuentes.	Las metodologías tradicionales suelen realizar las pruebas al final del ciclo de desarrollo, después de completar las fases de diseño e implementación, y se documentan de manera exhaustiva.
Equipo de desarrollo	Las metodologías ágiles se centran en equipos pequeños multifuncionales y autoorganizados, con roles más fluidos y con una colaboración estrecha entre todos los miembros.	Las metodologías tradicionales comúnmente utilizan equipos jerárquicos y especializados, con roles y responsabilidades definidos con menos colaboración entre diferentes disciplinas.

El examen de las características de las metodologías existentes permitió al equipo de desarrollo concluir que la opción más adecuada para el proyecto en cuestión es la adopción de una metodología ágil, debido a su capacidad de adaptarse y ser flexible ante los diversos escenarios que pueden surgir durante las diferentes fases del proceso de desarrollo. Además, las características analizadas relacionadas con el equipo de desarrollo muestran similitudes con la situación actual del proyecto.

Posterior a esta fase se realiza una comparación entre dos de las metodologías de desarrollo de enfoque ágil más utilizadas actualmente (XP, SCRUM y AUP) para realizar la selección basándose en la mayor cantidad de coincidencias con el escenario presente en el proyecto a desarrollar:

La metodología **XP (*Extreme Programming*)** es un enfoque ágil para el desarrollo de soft-ware que se centra en mejorar la calidad del software y la capacidad de respuesta a las necesidades cambiantes del cliente. XP se distingue por sus prácticas clave como la programación en parejas (*pair programming*), el desarrollo guiado por pruebas (*Test-Driven Development, TDD*), la integración continua, y la planificación de liberaciones frecuentes, lo que permite iteraciones rápidas y mejora continua en el proceso de desarrollo (Alice, 2021). Además, XP hace hincapié en la importancia de la comunicación constante y la colaboración entre todos los miembros del equipo, incluyendo a los clientes. Una de las filosofías centrales de XP es el concepto de

ritmo sostenible", que busca evitar la sobrecarga de trabajo y mantener un equilibrio saludable entre la vida laboral y personal de los desarrolladores.

La refactorización continua es otra característica importante de XP. En lugar de esperar hasta el final del desarrollo para hacer mejoras en el código, XP promueve la optimización constante del código para mantenerlo simple y fácil de mantener, sin cambiar su funcionalidad. En resumen, XP es una metodología ágil que combina prácticas como la programación en pares, el desarrollo guiado por pruebas, la refactorización continua y la comunicación continua para lograr un desarrollo de software más flexible y de alta calidad.

Por su parte, **Scrum** es una metodología ágil ampliamente utilizada para gestionar el desarrollo de software. Scrum se enfoca en la entrega incremental y la adaptabilidad mediante la organización del trabajo en ciclos llamados "*sprints*" (*Guía de Scrum* | *Guías de Scrum* 2024).

El proceso en Scrum se divide en sprints, que son períodos de tiempo cortos y fijos, generalmente de dos a cuatro semanas. Durante cada *sprint*, se completa un conjunto específico de funcionalidades o tareas, entregando una versión funcional del producto al final del ciclo. Esta estructura permite realizar entregas frecuentes y recibir retroalimentación continua. Scrum fomenta una entrega incremental y adaptativa, permitiendo ajustes rápidos a los requisitos y asegurando que el producto evolucione en función de las necesidades y *feedback* del cliente. La metodología promueve una alta colaboración y comunicación dentro del equipo, y busca mantener el enfoque en entregar valor continuo.

En resumen, Scrum es una metodología ágil que organiza el trabajo en *sprints*, involucra roles específicos como el *Product Owner* y *Scrum Master*, y utiliza ceremonias regulares para planificar, revisar y mejorar el desarrollo del software. Su enfoque en la entrega incremental y la adaptabilidad ayuda a asegurar que el producto final cumpla con las expectativas y necesidades cambiantes del cliente.

Finalmente, *The Agile Unified Process (AUP)* es una metodología de desarrollo ágil que adapta los principios del *Unified Process (UP)* para ser más flexible y adaptable a los cambios. AUP busca combinar las mejores prácticas del UP con un enfoque ágil para mejorar la eficiencia y la efectividad en el desarrollo de software (*The Agile Unified Process (AUP) Home Page* 2024).

AUP se basa en una serie de iteraciones y ciclos de desarrollo que permiten entregas incrementales del producto. Cada iteración incluye actividades de planificación, diseño, desarrollo, y pruebas, permitiendo que el software evolucione de manera continua y adaptativa. Esto permite a los equipos de desarrollo responder rápidamente a los cambios en los requisitos del cliente y ajustar el rumbo del proyecto según sea necesario. En resumen, AUP es una metodología que adapta el *Unified Process* a un enfoque ágil, organizando el trabajo en iteraciones y manteniendo las fases principales del proceso. Su enfoque en la flexibilidad, la entrega incremental y la documentación mínima permite una mejor adaptación a los cambios y una mayor eficiencia en el desarrollo de software.

Conociendo los elementos clave de estas metodologías, se procede a realizar una comparación detallada en función de diferentes aspectos relevantes para el proyecto:

Tabla 1.3. Comparación entre tipos de metodologías existentes

Aspectos	XP	SCRUM	AUP
Enfoque de Calidad	Alta calidad mediante desarrollo guiado por pruebas, refactorización continua, y programación en pares.	Calidad enfocada en entregas incrementales y revisión continua del producto al final de cada sprint.	Calidad mediante iteraciones y ciclos que incluyen actividades de pruebas y validación en cada fase del proceso.
Prácticas Clave	Desarrollo guiado por pruebas, refactorización continua, programación en pares, integración continua.	Sprints, daily stand-ups, revisión de sprint, retrospectivas de sprint, roles definidos (Product Owner, Scrum Master, equipo de desarrollo).	Iteraciones, fases del proceso (Inception, Elaboration, Construction, Transition), roles y responsabilidades, documentación mínima.
Flexibilidad	Alta flexibilidad en cuanto a cambios durante el desarrollo, con un enfoque en adaptarse a los cambios en los requisitos.	Alta flexibilidad mediante ajustes durante los sprints y adaptaciones basadas en la retroalimentación del cliente.	Flexibilidad moderada, adaptando el proceso del Unified Process a un enfoque ágil, con iteraciones que permiten ajustes.
Comunicación	Comunicación constante entre desarrolladores, y entre el equipo y el cliente, facilitada por prácticas como la programación en pares y la integración continua.	Comunicación diaria en las reuniones diarias (stand-ups), y retroalimentación continua durante los sprints y revisiones.	Comunicación estructurada en cada fase y durante las iteraciones, con un enfoque en la colaboración entre roles definidos.
Entregas	Entregas frecuentes de versiones funcionales del software al final de cada iteración, con una alta frecuencia de integración y despliegue.	Entregas incrementales al final de cada sprint, con un incremento funcional del producto en cada ciclo.	Entregas al final de cada iteración dentro de las fases del proceso, con un enfoque en entregar valor continuo y ajustado a los cambios.

El equipo, en colaboración con el cliente, eligió la metodología XP (*Extreme Programming*) para el desarrollo del proyecto debido a su flexibilidad y capacidad para adaptarse a cambios. XP permite una mayor autonomía al equipo y ofrece entregas continuas de software de alta calidad con iteraciones más cortas que las de Scrum, lo que se ajusta mejor a las necesidades del proyecto para una implementación rápida. Aunque

Scrum y el *Agile Unified Process (AUP)* también eran opciones, XP proporciona un enfoque más intensivo y detallado en asegurar la calidad del código durante el desarrollo y una rápida respuesta a cambios durante el desarrollo.

1.3.2. Herramientas Case

Las herramientas CASE (*Computer-Aided Software Engineering*) se definen como aplicaciones de software diseñadas para apoyar las diversas actividades en el proceso de desarrollo de software (Pressman y Maxim, 2020). Estas herramientas están orientadas a mejorar la productividad, la calidad y la eficiencia en la ingeniería de software al automatizar tareas repetitivas y proporcionar soporte en áreas como el diseño, la codificación, la prueba y la gestión de proyectos.

En un análisis realizado en conjunto con el cliente se tomó la decisión de utilizar como herramienta CASE para el proyecto el software Visual paradigm producto de la existencia de un conocimiento previo por parte del equipo de desarrollo de esta tecnología durante su trabajo en anteriores proyectos:

Visual Paradigm (v8.0):

Visual Paradigm es una herramienta integral de modelado y diseño de software que facilita la creación de diagramas UML (*Unified Modeling Language*) y otros modelos esenciales para el desarrollo de sistemas (*Visual Paradigm Reviews, Pricing and Features* 2024). La herramienta está diseñada para apoyar la ingeniería de software a través de una interfaz intuitiva que permite a los usuarios generar, gestionar y documentar modelos de manera eficiente.

La elección de Visual Paradigm se basó en varios criterios clave:

- **Modelado UML:** Visual Paradigm proporciona soporte para una amplia gama de diagramas UML, incluyendo diagramas de clases, casos de uso, secuencia y actividad, entre otros.
- **Diseño Visual:** La herramienta ofrece un entorno visual que facilita la creación y edición de diagramas, con funciones de arrastrar y soltar y una variedad de plantillas.
- **Generación de Código:** Permite la generación automática de código a partir de los modelos UML y también soporta la ingeniería inversa para crear diagramas a partir de código existente.
- **Documentación y Reportes:** Facilita la generación de documentación detallada y reportes basados en los modelos, ayudando en la comunicación y presentación de diseños.
- **Integración y Colaboración:** Ofrece capacidades de integración con otras herramientas y soporta la colaboración entre miembros del equipo mediante el uso compartido de modelos y diagramas.

1.3.3. Lenguajes de programación

Para seleccionar el lenguaje a utilizar en los apartados de frontend y backend del sistema se realizó un estudio de las tecnologías más utilizadas actualmente y realizando una comparación entre estas de sus características principales para validar la decisión final.

Backend:

Este segmento de la aplicación se encarga de manejar tanto los datos como la lógica empresarial. Su estructura puede variar, pero generalmente incluye servidores, APIs (Interfaz de Programación de Aplicaciones) y bases de datos, los cuales aseguran el funcionamiento continuo del sistema. Durante el proceso de análisis para seleccionar el lenguaje de programación más adecuado para el proyecto, el equipo llevó a cabo una evaluación detallada de tres de las opciones más comunes en este ámbito: Java, Python con Django y JavaScript con Node.js.

Python (v3.12):

Python es un lenguaje de programación versátil y accesible, ideal para el desarrollo web con Django (*Welcome to Python.org* 2024). Su sintaxis clara y legible facilita tanto el aprendizaje como su implementación, convirtiéndose en una excelente opción incluso para quienes están iniciándose en el mundo de la programación.

Este lenguaje se integra perfectamente con Django, un potente framework diseñado para el desarrollo web. Juntos, Python y Django permiten construir aplicaciones web robustas y eficientes. Django aprovecha Python para gestionar la lógica del backend, manejar bases de datos y procesar las solicitudes de los usuarios, proporcionando una estructura organizada y reutilizable que simplifica el desarrollo.

En resumen, Python es la base que impulsa a Django, ofreciendo una plataforma estable y flexible para crear aplicaciones web funcionales y escalables. Esta combinación resulta ideal para desarrollar desde sitios web básicos hasta soluciones complejas con facilidad y eficiencia.

Java (v.18):

Java es un lenguaje de programación orientado a objetos que se caracteriza por su portabilidad, lo que significa que el código escrito en Java puede ejecutarse en cualquier dispositivo que tenga una máquina virtual Java (JVM). Desarrollado por Sun Microsystems y lanzado en 1995, Java es ampliamente utilizado en diversas áreas, desde aplicaciones empresariales hasta desarrollo de aplicaciones móviles y sistemas embebidos. Su popularidad se debe a su robustez, seguridad y a su rica biblioteca de clases que facilita el desarrollo de software escalable y eficiente (*Java (programming language)* 2024).

JavaScript (ES2023):

JavaScript es un lenguaje de programación dinámico y versátil que se utiliza principalmente para desarrollar aplicaciones web interactivas (*JavaScript | MDN* 2024). Su naturaleza basada en prototipos le confiere una flexibilidad única para manejar objetos y herencia, lo que amplía su aplicabilidad en una gran variedad de escenarios de desarrollo.

Uno de los aspectos más destacados de JavaScript es su capacidad para gestionar eventos y operaciones asíncronas, convirtiéndolo en una opción ideal para construir interfaces de usuario reactivas y aplicaciones que requieren interacción en tiempo real.

En resumen, JavaScript es una herramienta esencial para cualquier desarrollador web. Su versatilidad para ser empleado tanto en el frontend como en el backend lo ha consolidado como un pilar fundamental en el desarrollo de aplicaciones modernas.

Tabla 1.4. Comparación entre los lenguajes para apartado del Backend

Aspectos	Java	Python	JavaScript
Marcos de Trabajo (Frameworks)	- Frameworks robustos como Spring, Hibernate, y JavaServer Faces (JSF).	- Frameworks como Django, Flask, y FastAPI ofrecen capacidades rápidas y flexibles para el desarrollo web.	- Frameworks para backend como Express.js para Node.js.
Trabajo con base de datos	- JDBC para conectividad a bases de datos relacionales. - Soporte amplio para SQL y NoSQL a través de bibliotecas y frameworks (Hibernate, JPA).	- Soporte fuerte para SQL y NoSQL. - ORMs populares como SQLAlchemy, Django ORM.	- Conexiones a bases de datos relacionales y NoSQL a través de bibliotecas (e.g., Sequelize para SQL, Mongoose para MongoDB).
Desempeño	- Generalmente rápido y eficiente, especialmente en aplicaciones empresariales grandes. - Compilado, lo que puede mejorar el rendimiento.	- Menos rápido en comparación con Java para ciertas aplicaciones, pero generalmente adecuado para muchas aplicaciones web. - Interpretado, lo que puede afectar el rendimiento en comparación con los lenguajes compilados.	- Bueno para aplicaciones en tiempo real debido a su naturaleza asincrónica. - Interpretado, pero el uso de motores V8 y Node.js puede ofrecer un rendimiento competitivo.
Popularidad en el Desarrollo Web	- Amplio uso en aplicaciones empresariales, sistemas grandes y aplicaciones web a gran escala.	- Muy popular para desarrollo rápido de aplicaciones web y prototipos. - Amplia comunidad y soporte.	- Dominante en el desarrollo frontend con frameworks como React y Angular. - Node.js ha incrementado su uso para backend.

El equipo de desarrollo, en colaboración con el cliente, decidió seleccionar Python como el lenguaje de programación para el backend debido a su sintaxis simple, que facilita un aprendizaje más rápido en comparación con Java. A pesar de las ventajas que ofrece JavaScript con Node.js, que también fue considerado por su capacidad para manejar aplicaciones de alto rendimiento y su flexibilidad en el desarrollo web, el equipo optó por Python. La decisión se basa en la facilidad de uso de Python y su capacidad para acelerar el proceso de desarrollo, haciendo que sea más adecuado para los objetivos y plazos del proyecto en cuestión.

Frontend:

El componente frontend de una aplicación se refiere a la sección encargada de la interfaz gráfica de usuario (GUI), la cual permite a los usuarios interactuar directamente con la aplicación. Este segmento abarca diversos elementos visuales, tales como botones, cuadros de selección, gráficos y textos, que facilitan la interacción del usuario con la aplicación.

Los navegadores web, utilizados para acceder a servicios en línea, incorporan lenguajes específicos diseñados para el desarrollo del frontend. Esta comprensión permitió al equipo de desarrollo optar por los siguientes lenguajes para esta parte del proyecto:

JavaScript (ES2023):

JavaScript, cuando se utiliza en el desarrollo frontend, se convierte en una herramienta poderosa para construir interfaces de usuario dinámicas y reactivas. Es esencial para crear aplicaciones web interactivas debido a su capacidad para manipular el contenido del navegador en tiempo real y responder a las acciones del usuario (ibíd.).

Mediante bibliotecas y frameworks como React, JavaScript permite construir componentes reutilizables y gestionar el estado de la aplicación de manera eficiente. React, en particular, facilita la creación de interfaces de usuario mediante su enfoque basado en componentes y su eficiente sistema de actualización del DOM (*Document Object Model*), lo que resulta en aplicaciones más rápidas y fáciles de mantener. Esta capacidad de manipulación dinámica del DOM y la interacción en tiempo real son lo que hacen de JavaScript una herramienta tan versátil y potente para el desarrollo frontend (ibíd.).

Además de JavaScript y React.js, se utilizarán HTML y CSS los cuales son lenguajes de marcado y de estilo para estructurar y diseñar las interfaces de sistema.

HTML (v.5.2):

HTML proporciona una estructura para el contenido de la web mediante el uso de etiquetas que definen distintos tipos de información. Las etiquetas HTML, como `<header>`, `<footer>`, `<article>`, y `<section>`, permiten organizar el contenido de manera semántica y lógica, facilitando tanto la comprensión por parte de los desarrolladores como la accesibilidad para los usuarios y los motores de búsqueda (HTML 2024).

Asimismo, HTML permite la incorporación de diversos tipos de medios en las páginas web, como imágenes, videos y audios, a través de etiquetas específicas como `<img>`, `<video>`, y `<audio>`. Esto proporciona a los desarrolladores la capacidad de enriquecer el contenido web y ofrecer una experiencia multimedia más completa a los usuarios

CSS (v3):

CSS (*Cascading Style Sheets*) se utiliza para aplicar estilos visuales a los elementos HTML, como colores, fuentes, márgenes, y alineaciones. Mediante la separación de la estructura (HTML) del diseño (CSS), los desarrolladores pueden cambiar la apariencia de una página web sin alterar su contenido, permitiendo un diseño más flexible y mantenible (CSS 2024).

Además, CSS es fundamental para el diseño responsivo, asegurando que las páginas web se adapten a diferentes tamaños de pantalla y dispositivos. Mediante el uso de media queries, los desarrolladores pueden aplicar estilos específicos para entornos variados, como teléfonos móviles, tabletas y computadoras de escritorio, mejorando significativamente la experiencia del usuario.

1.3.4. Marcos de Trabajo

En este apartado se procedió en la realización de un estudio para poder definir las tecnologías a utilizar como marcos de trabajo para desarrollar el sistema analizando características de los más utilizados en la actualidad para las aplicaciones web.

Backend:

Durante el proceso de selección del Marco de Trabajo para el apartado del backend se realizó una comparación entre los datos de uno de los más utilizados pertenecientes al lenguaje Python seleccionado: Django, FastAPI y Flask:

FastAPI (v.0.111.1): Es un framework innovador para construir APIs en Python, creado por Sebastián Ramírez. Su propósito principal es ofrecer una solución rápida y eficiente para desarrollar aplicaciones web y APIs robustas (*FastAPI - FastAPI* 2024).

Una de las grandes fortalezas de FastAPI es su rendimiento sobresaliente. Gracias a su arquitectura basada en Starlette y Pydantic, proporciona una velocidad comparable a la de lenguajes como Node.js o Go, haciendo que las aplicaciones respondan de manera ágil y eficiente. Otra ventaja significativa es su capacidad para generar documentación interactiva automáticamente. Al utilizar OpenAPI y JSON Schema, FastAPI crea una interfaz web donde los desarrolladores pueden explorar y probar las APIs de forma intuitiva, facilitando el proceso de desarrollo y prueba. Además, el framework se destaca por su uso del tipado estático de Python, lo que no solo mejora la claridad del código, sino que también asegura una validación precisa y una detección temprana de errores. En resumen, FastAPI combina rapidez, eficiencia y una experiencia de desarrollo simplificada, permitiendo a los desarrolladores construir APIs de alto rendimiento con menos esfuerzo y en menos tiempo.

Django (v.4.2.2): Django es un framework web de alto nivel basado en Python que fomenta el desarrollo rápido y un diseño limpio y pragmático. Creado por desarrolladores experimentados, se encarga de gran parte de las complicaciones del desarrollo web, para que puedas concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda. Es gratuito y de código abierto (*Django* 2024).

Algunos aspectos clave de Django, como se describe en su documentación oficial, incluyen:

- **Ridículamente rápido:** Django fue diseñado para ayudar a los desarrolladores a llevar las aplicaciones desde el concepto hasta su finalización lo más rápido posible.
- **Tranquilizadoramente seguro:** Django se toma la seguridad en serio y ayuda a los desarrolladores a evitar muchos errores de seguridad comunes.
- **Extremadamente escalable:** Algunos de los sitios más activos de la web aprovechan la capacidad de Django para escalar de forma rápida y flexible.
- **Completamente cargado:** Django incluye docenas de extras que puedes usar para manejar tareas comunes de desarrollo web. Django se encarga de la autenticación de usuarios, la administración de contenido, los mapas del sitio, los feeds RSS y muchas otras tareas.
- **Increíblemente versátil:** Las empresas, organizaciones y gobiernos han utilizado Django para crear

todo tipo de cosas: desde sistemas de gestión de contenidos hasta redes sociales y plataformas informáticas científicas.

Flask (v.3.0.3): Es un micro framework web escrito en Python. Se clasifica como un micro framework porque no requiere herramientas o bibliotecas particulares. No tiene una capa de abstracción de base de datos, validación de formularios ni ningún otro componente donde las bibliotecas de terceros preexistentes proporcionen funciones comunes. Sin embargo, Flask admite extensiones que pueden agregar características de la aplicación como si estuvieran implementadas en Flask mismo. Existen extensiones para mapeadores relacionales de objetos, validación de formularios, manejo de carga, varias tecnologías de autenticación abierta y varias herramientas comunes relacionadas con el framework (*Flask web framework* 2024).

Tras revisar los datos previamente presentados, se llevó a cabo una evaluación comparativa entre los diferentes marcos de trabajo para validar la elección de Django como el marco más adecuado para el desarrollo de la aplicación. A continuación, se detallan los diversos aspectos que fueron analizados:

Tabla 1.5. Comparación de marcos de trabajo de apartado del backend

Aspectos	FastAPI	Django	Flask
Velocidad de desarrollo	Rápida: La generación automática de documentación y el uso de tipado estático permiten una rápida implementación y pruebas.	Muy Rápida: La arquitectura "baterías incluidas" permite desarrollar aplicaciones completas sin necesidad de integrar componentes externos, acelerando el proceso de desarrollo.	Rápida: Su enfoque minimalista y la ausencia de componentes innecesarios permiten una rápida puesta en marcha, aunque requiere más configuración manual para funcionalidades avanzadas.
Rendimiento	Alto: Diseñado para ser extremadamente rápido y eficiente, con tiempos de respuesta comparables a los de lenguajes como Node.js.	Consistente y Sólido: Rendimiento robusto, ideal para aplicaciones grandes y complejas, con herramientas integradas que optimizan la gestión de datos y la escalabilidad.	Alto: Ligeroy eficiente, pero el rendimiento puede depender de la configuración y extensiones utilizadas.
Curva de aprendizaje	Baja: La simplicidad y la documentación extensa facilitan el aprendizaje. El uso de Python y el tipado estático ayudan a reducir errores.	Baja para el equipo de desarrollo: Con experiencia previa en Django, la curva de aprendizaje es más suave. La abundancia de recursos y la documentación extensa nos permiten utilizarlo de manera más eficiente.	Baja: Fácil de aprender debido a su simplicidad, pero su flexibilidad requiere que los desarrolladores construyan manualmente características que otros marcos ya incluyen.
Comunidad y ecosistema	Creciente: Relativamente nueva, pero con una comunidad en expansión y un ecosistema en desarrollo.	Amplia y madura: Una de las comunidades más grandes y activas en el ecosistema Python, con muchos recursos, tutoriales y paquetes disponibles, y un ecosistema probado en producción.	Amplia: Establecida y activa, con una gran cantidad de extensiones y tutoriales disponibles, aunque no tan robusta como la de Django.
Escalabilidad	Alta: Diseñado para manejar aplicaciones de alta demanda y escalabilidad sin sacrificar el rendimiento.	Muy Alta: Ideal para aplicaciones grandes y complejas; diseñado para escalar con herramientas integradas de optimización y gestión de datos, garantizando un rendimiento sólido en aplicaciones de gran escala.	Moderada a Alta: Adecuado para aplicaciones pequeñas a medianas, y puede escalar con la adición de herramientas y extensiones específicas.
Nelson Castroman Aguiar		17	Gabriela de la Caridad Rodríguez Rey

El marco de trabajo de Django ofrece una estructura sólida y características maduras que son altamente valiosas para el desarrollo de aplicaciones web completas. Aunque la complejidad puede ser mayor en comparación con otros frameworks, nuestra experiencia previa con Django reduce significativamente la curva de aprendizaje, permitiéndonos aprovechar su riqueza en funcionalidades y robustez del ecosistema de manera más eficiente. A pesar de las capacidades de FastAPI y Flask, la combinación de nuestra experiencia con Django y sus herramientas integradas hacen de este marco la opción más adecuada para cumplir con los requisitos del proyecto de manera óptima.

Frontend:

La elección de un marco de trabajo para el desarrollo frontend de un proyecto se basa en la evaluación de los requisitos específicos de la aplicación. Para cumplir con este criterio, se llevó a cabo un análisis de los frameworks más populares y utilizados en esta área (*Encuesta para desarrolladores de Stack Overflow 2024* 2024).

- **React:** Desarrollado por Facebook, es una biblioteca para construir interfaces de usuario. Permite crear aplicaciones web interactivas y eficientes mediante componentes reutilizables y un enfoque basado en el estado (*React* 2024).
- **Angular:** Mantenido por Google, es un marco de trabajo completo que proporciona una solución robusta para construir aplicaciones web de una sola página (SPA). Angular ofrece un conjunto completo de herramientas y características, como enlace bidireccional de datos e inyección de dependencias (*Angular* 2024).
- **Vue.js:** Es un marco progresivo que se puede adoptar de manera incremental. Vue es conocido por su simplicidad y flexibilidad, permitiendo a los desarrolladores agregar capacidades avanzadas a sus aplicaciones de manera gradual (*Vue.js* 2024).

Tabla 1.6. Comparación de Marcos de trabajo del apartado del frontend

Aspectos	React	Angular	Vue.js
Curva de Aprendizaje	Moderada: Aprender React requiere familiarizarse con JSX, que combina JavaScript con HTML, y entender el manejo del estado con herramientas como Redux o Context API. Aunque tiene una curva de aprendizaje, es menos empinada en comparación con frameworks más complejos.	Empinada: Angular tiene una curva de aprendizaje pronunciada debido a su arquitectura y conceptos avanzados, como la inyección de dependencias, el enlace bidireccional de datos y una sintaxis compleja. Esto puede requerir más tiempo para dominar.	Suave: Vue.js es reconocido por su simplicidad y facilidad de aprendizaje. Su documentación clara y estructura intuitiva permiten a los desarrolladores aprender rápidamente y comenzar a construir aplicaciones con menos fricción.
Rendimiento	Alto: React.js utiliza un Virtual DOM para gestionar las actualizaciones de la interfaz de usuario de manera eficiente. Esto reduce el costo de las actualizaciones del DOM real, ofreciendo un rendimiento sólido y fluido en aplicaciones dinámicas.	Bueno: Aunque Angular.js también ofrece un buen rendimiento, su tamaño y la complejidad del framework pueden llevar a un rendimiento algo más lento en comparación con React y Vue, especialmente en aplicaciones grandes.	Muy alto: Vue.js está diseñado para ofrecer un rendimiento excepcional con un sistema de reactividad eficiente. Su enfoque en la actualización de datos de manera eficiente asegura un rendimiento rápido y reactivo en aplicaciones interactivas.
Flexibilidad	Alta: React.js se destaca por su flexibilidad, permitiendo a los desarrolladores integrar diversas bibliotecas y herramientas según las necesidades del proyecto. Esto lo hace adecuado para una amplia variedad de casos de uso y arquitecturas.	Baja: Angular.js proporciona un enfoque más estructurado y menos flexible debido a su diseño integral. Esto puede ser una ventaja para aplicaciones grandes y bien definidas, pero limita la flexibilidad en comparación con soluciones más modulares.	Alta: Vue.js ofrece una gran flexibilidad, permitiendo a los desarrolladores adoptar el framework de manera gradual. Puede ser utilizado tanto para proyectos pequeños como para aplicaciones complejas, y se adapta bien a diferentes estilos de desarrollo.

Aunque los tres marcos evaluados ofrecen una variedad de características comunes, el enfoque modular y la curva de aprendizaje relativamente accesible de React.js se presentan como la alternativa más adecuada para el equipo de desarrollo.

1.3.5. Gestión de Base de Datos

Un sistema de gestión de bases de datos (SGBD) es un software diseñado para facilitar la creación, modificación y administración de bases de datos. Estos sistemas son esenciales para almacenar grandes volúmenes de información de forma estructurada y eficiente, además de ofrecer herramientas para la recuperación y manipulación de datos. Los SGBD se dividen en varios tipos, como los sistemas de gestión de bases de datos relacionales (RDBMS), que organizan la información en tablas estructuradas. También existen sistemas no relacionales, que manejan datos no estructurados o semiestructurados, tales como documentos o datos provenientes de sensores (Silberschatz; Korth y Sudarshan, 2020).

A continuación, se realiza una comparación entre tres de los principales gestores de bases de datos con el objetivo de presentar un mayor entendimiento sobre el tema:

Tabla 1.7. Comparación de Gestores de bases de datos

Aspectos	MySQL	PgAdmin (para PostgreSQL)	SQLite
Modelos de datos y tipos de datos	Relacional. Soporta tipos estándar (int, varchar, date) y tipos específicos (JSON, spatial).	Relacional. Depende del PostgreSQL; soporte extensivo para tipos avanzados (JSON, hstore, arrays).	Relacional. Tipos estándar básicos (INTEGER, TEXT, REAL) y tipos dinámicos (NULL).
Rendimiento	Alto rendimiento, optimizado para lectura y escritura con índices avanzados.	Depende del PostgreSQL; generalmente alto rendimiento con soporte para optimización avanzada.	Bueno para aplicaciones ligeras; rendimiento adecuado para aplicaciones pequeñas a medianas.
Escalabilidad	Buena escalabilidad vertical y horizontal, soporta clustering y replicación.	Escalabilidad dependiente de PostgreSQL; soporta clustering y particionamiento.	Limitado; generalmente usado en aplicaciones locales o pequeñas.
Compatibilidad con No SQL	Limitada. Soporta JSON para algunos casos de uso NoSQL, pero no es una base de datos NoSQL completa.	Soporte robusto para JSON y datos no estructurados debido a la flexibilidad de PostgreSQL.	No compatible. Estrictamente relacional sin soporte para características NoSQL.

El sistema gestor de bases de datos elegido para el proyecto es PgAdmin para PostgreSQL, un sistema con el cual el equipo de desarrollo ya tiene experiencia previa, lo que evita la necesidad de aprender una nueva tecnología y optimiza el tiempo de desarrollo (*pgAdmin User Management in Server Mode* | EDB 2024).

PgAdmin (v.6.1):

PgAdmin 4 es un gestor de bases de datos de código abierto diseñado específicamente para trabajar con el sistema de gestión de bases de datos PostgreSQL. Proporciona una interfaz gráfica intuitiva y completa para administrar y realizar tareas de mantenimiento en bases de datos PostgreSQL (*pgAdmin User Management in Server Mode* 2024).

Algunas características clave del gestor de bases de datos PgAdmin 4 son:

- **Interfaz gráfica intuitiva:** PgAdmin 4 ofrece una interfaz gráfica fácil de usar que permite a los usuarios administrar y trabajar con bases de datos PostgreSQL de manera eficiente. La interfaz proporciona una navegación clara y una organización lógica de los objetos de la base de datos, como tablas, vistas, funciones, esquemas, entre otros.
- **Administración de bases de datos:** PgAdmin 4 permite realizar tareas de administración esenciales, como crear y eliminar bases de datos, crear y modificar esquemas, gestionar roles y permisos, crear copias de seguridad y restaurar bases de datos, y realizar monitorización y optimización del rendimiento.
- **Editor de consultas SQL:** PgAdmin 4 incluye un editor de consultas SQL integrado que facilita la escritura, ejecución y depuración de consultas SQL. Este editor ofrece resaltado de sintaxis, autocompletado de código, sugerencias de objetos y funciones, y herramientas para el análisis y la optimización de consultas.
- **Explorador de objetos:** PgAdmin 4 proporciona un explorador de objetos de base de datos que permite navegar y manipular fácilmente los objetos de la base de datos, como tablas, vistas, funciones, índices, restricciones y secuencias. También ofrece la capacidad de ver y editar datos en las tablas directamente desde la interfaz.
- **Herramientas de visualización de datos:** PgAdmin 4 ofrece herramientas visuales para explorar y visualizar datos almacenados en bases de datos PostgreSQL. Esto incluye la capacidad de crear gráficos, diagramas ER (Entidad-Relación) y vistas personalizadas para analizar y presentar los datos de manera efectiva.
- **Soporte para extensiones y complementos:** PgAdmin 4 permite extender su funcionalidad mediante la instalación de extensiones y complementos. Hay una variedad de complementos disponibles desarrollados por la comunidad que agregan características adicionales al gestor de bases de datos.
- **Soporte multiplataforma:** PgAdmin 4 es compatible con varios sistemas operativos, incluyendo Windows, macOS y Linux, lo que brinda flexibilidad para su uso en diferentes entornos de desarrollo.

1.3.6. Editor de código

Un editor de código es una aplicación de software diseñada para facilitar la escritura y edición de texto, especialmente el código fuente de programas informáticos. Estos editores suelen ofrecer características como resaltado de sintaxis, autocompletado y la capacidad de instalar complementos que amplían sus funcionalidades. Ejemplos conocidos de editores de código son Visual Studio Code, Sublime Text, Atom y Vim. Estas herramientas son esenciales para los programadores, ya que permiten manejar el código de manera eficiente, lo cual es crucial para el desarrollo de software y sistemas informáticos. En base a la experiencia previa con Visual Studio Code, los desarrolladores optaron por emplear este editor para el desarrollo de la aplicación en cuestión (*Visual Studio Code - Code Editing. Redefined* 2024).

Visual Studio Code (v1.87.2):

Visual Studio Code es un editor de código fuente gratuito y eficiente, diseñado para el desarrollo y depuración de aplicaciones web y en la nube modernas. Es compatible con los sistemas operativos Windows, Linux y macOS. Entre sus características destacan la búsqueda integrada, la opción de renombrar funciones en todas sus instancias, y la capacidad de ajustar configuraciones para mostrar errores. Además, Visual Studio Code ofrece soporte para la depuración y una integración completa con Git. El editor permite la instalación de extensiones para añadir nuevos lenguajes, temas, depuradores y conectar con servicios adicionales. Con más de 30,000 extensiones y temas disponibles en su sitio oficial, los usuarios pueden personalizar tanto la funcionalidad como la apariencia del editor para satisfacer sus necesidades, preferencias y estilo personal (*Visual Studio Code - Code Editing. Redefined* 2024).

Conclusiones Parciales

El análisis de los elementos y aspectos abordados en el presente capítulo proporciona varias conclusiones clave sobre el software a desarrollar, según lo documentado:

- La definición de términos esenciales como gestión de servicios, gestión administrativa, entre otros es crucial para una comprensión más detallada del proyecto.
- A partir de un análisis detallado, se seleccionaron como herramientas y tecnologías fundamentales para el desarrollo del sistema los lenguajes de programación Python v3.11.5 y JavaScript vES13, así como las tecnologías Django v4.2.6, React v18.3.1 y el gestor de base de datos MySQL v5.2.1.
- El análisis de los softwares homólogos ha brindado una visión general sobre los estándares y requisitos esenciales para un sistema enfocado en la gestión de matrículas y distribución en instituciones educativas. Además, ha resaltado las características más comunes y menos frecuentes en estos sistemas.
- La selección de las tecnologías para el desarrollo en el backend y frontend (React.js y Django) optimiza el acceso a la información requerida por el cliente, reduciendo al mismo tiempo el tiempo necesario para la capacitación del equipo de desarrollo.

Análisis y diseño de la solución

En este capítulo, se ofrece una completa descripción del sistema propuesto, el cual se ha concebido siguiendo las fases de Exploración y Planificación de la metodología XP para el desarrollo de software. Aquí, se exploran en detalle las características esenciales del sistema, incluyendo los diversos artefactos generados durante cada fase del proceso. Las historias de usuario también hacen su aparición, estas narran las necesidades del sistema y funcionan como guía fundamental para orientar su desarrollo. Se realiza una estimación del tiempo requerido para llevar a cabo el proyecto y se establece la arquitectura del sistema junto con el diseño de las clases correspondientes. Asimismo, en este capítulo quedan establecidos los patrones de diseño, el patrón arquitectónico y las tarjetas de clases, responsabilidad y colaboración (CRC).

2.1. Descripción del sistema a desarrollar

Con el fin de cumplir los objetivos planteados y en respuesta a las necesidades actuales, se propone el desarrollo de un sistema que gestione el proceso de ingreso a la universidad. Este sistema permitirá la distribución equitativa de los estudiantes en grupos y facultades, tomando en cuenta su carrera y tipo de curso. Además, incluirá un algoritmo de inteligencia artificial diseñado para asegurar una asignación justa y lo más aleatoria posible. El sistema también permitirá la generación automatizada de reportes, la gestión de usuarios y la asignación de roles, así como la toma de decisiones informadas. El sistema gestiona la información de los nuevos estudiantes, el sistema permite crear un nuevo curso académico al cual pertenecerán los estudiantes una vez que sean importados. El sistema ofrece la funcionalidad de administrar cursos, facultades, carreras, grupos y de configurar la distribución. El usuario podrá distribuir los estudiantes por facultades y después por grupos de forma equitativa y aleatoria, teniendo en cuenta criterios como carrera, sexo, provincia de origen y promedio académico. El usuario también tendrá la capacidad de gestionar toda la información relacionada con los estudiantes y todo el proceso de distribución de estos. El usuario con el rol de administrador podrá gestionar toda la información relacionada con los usuarios, asignar roles.

En la siguiente imagen se muestra un esquema general para una mejor comprensión del sistema a desarrollar.2.1.

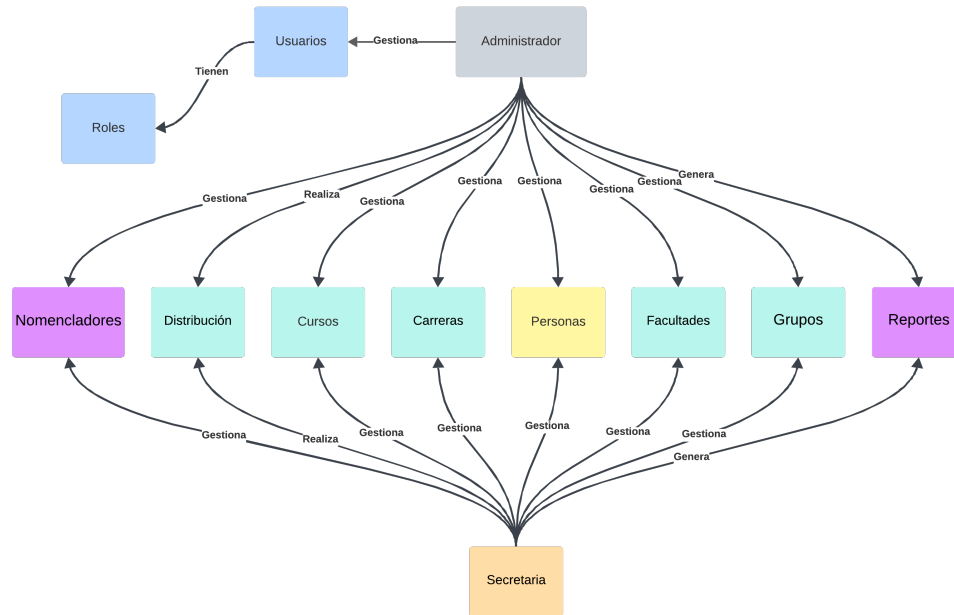


Figura 2.1. Mapa de Navegación

2.2. Relación de usuarios con el sistema

Tabla 2.1. Relación de usuarios con el sistema

Usuarios	Descripción
Usuario autenticado y rol Administrador	Es el encargado de gestionar las personas, curso académico, facultades, carreras y grupos, configurar y realizar la distribución por facultad, distribución por grupos, gestionar las funciones de seguridad del sistema, permitir a usuarios el acceso al sistema, asignar un rol al usuario, generar reportes.
Usuario autenticado y rol Secretaria	Es el encargado de gestionar las personas, curso académico, facultades, carreras y grupos, configurar y realizar la distribución por facultad, distribución por grupos, realizar reportes.

2.3. Fase de exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos

meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología (Letelier y PenadÃ©s, 2021).

2.3.1. Requerimientos no funcionales

Los requerimientos no funcionales, como su nombre sugieren, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema (Molina Hernández; Granda Dihigo y Velázquez Cintra, 2019).

Requisitos no funcionales de usabilidad RNF1. El sistema debe permitir la fácil interacción con el mismo para obtener de manera rápida y efectiva a la información deseada. También, debe ser una interfaz de manejo cómodo que permita a los usuarios sin experiencia una rápida adaptación.

Requisitos no funcionales de software

RNF2.

- Para el cliente se debe utilizar el:
 - Navegador web Mozilla Firefox 95.0 o superior.
 - Navegador web Google Chrome 96.0 o superior.
 - Otros navegadores actualizados.
- Para el servidor se debe utilizar:
 - Python en su versión 3.12 o superior.
 - PgAdmin 6.1 o superior.

Requisitos no funcionales de hardware

RNF3. Requerimientos mínimos para un funcionamiento adecuado:

- Intel Core i3 o superior, así como su equivalente en versiones AMD.
- Disco duro con capacidad de 160 GB como mínimo.
- Memoria RAM 4 GB o superior.
- Tarjetas de red cableada, inalámbrica o modem.

Requisitos no funcionales de seguridad

RNF4. Se realizarán validaciones a la información contenida en el sistema. El administrador, y secretarías, se categorizarán como usuarios, todos deben de estar autenticados para ver la información referente a su rol, según lo definido en el sistema.

Requisitos no funcionales de portabilidad

RNF5. El sistema debe brindar la posibilidad de ser ejecutado en distintos sistemas operativos como Windows, Linux y Mac Os.

Requisitos no funcionales de rendimiento**RNF6.** Tiempo de Respuesta:

- El sistema debe responder a las solicitudes de los usuarios en menos de 2 segundos para operaciones estándar, como la navegación entre secciones o la carga de datos.
- Las operaciones complejas, como la ejecución del algoritmo genético, deben completarse en un tiempo máximo de 5 segundos.

Requisitos no funcionales de mantenibilidad RNF7. El sistema debe contar con documentación interna en el código, utilizando comentarios claros y consistentes que expliquen la lógica de cada módulo y función. Esto facilita la comprensión y modificación del código por otros desarrolladores en el futuro.

2.3.2. Historias de usuario

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento las historias de usuario pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas (Letelier y PenadÃ©s, 2021).

A continuación, se muestran 3 del total de historias de usuario:

Tabla 2.2. Historia de usuario # 1

Historia de usuario	
Número: 1	Nombre: Gestionar usuario
Usuario: Administrador	
Prioridad en negocio: Alta	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al administrador agregar un usuario a partir de los campos nombre, usuario, rol y contraseña. Además debe permitir editar, eliminar, buscar y ver el listado de todos los usuarios	
Observaciones: El usuario con rol Administrador debe estar previamente autenticado	

Tabla 2.3. Historia de usuario # 2

Historia de usuario	
Número: 2	Nombre: Autenticar usuario
Usuario: Administrador, Secretaria	
Prioridad en negocio: Alta	Riesgo en desarrollo: N/A

Continúa en la próxima página

Tabla 2.3. Continuación de la página anterior

Puntos estimados: 1	Iteración asignada: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al usuario autenticarse en el sistema con su usuario y su contraseña	
Observaciones: En caso de no ser un usuario con acceso al sistema, no podrá autenticarse.	

Tabla 2.4. Historia de usuario # 3

Historia de usuario	
Número: 3	Nombre: Gestionar persona
Usuario: Administrador, Secretaria	
Prioridad en negocio: Media	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 1
Programador responsable: Gabriela de la C. Rodríguez Rey	
Descripción: El sistema debe permitir al usuario importar un listado de personas a partir de un archivo excel. Además debe permitir editar, eliminar, buscar, ver el listado de todas las personas y exportar una plantilla vacía en formato excel.	
Observaciones: El usuario debe estar previamente autenticado.	

El resto de las historias de usuario se muestran en el apéndice A de los anexos.

2.4. Fase de planificación

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la ?velocidad? de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración (Letelier y PenadÃ©s, 2021).

2.4.1. Estimación de esfuerzo por Historia de Usuario

Tras la definición de las Historias de Usuario (HU), el equipo de desarrollo procede a realizar una planificación del tiempo requerido para su implementación, lo que permite determinar la duración total del proceso de desarrollo de la aplicación. En el contexto de la investigación actual, la estimación del esfuerzo por HU se traduce en un período total aproximado de 8.9 semanas.

2.4.2. Plan de iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fueren la creación de esta arquitectura, sin embargo, esto no siempre es posible debido a que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción (Letelier y PenadÃ©s, 2021).

2.4.3. Plan de duración de las iteraciones

Tabla 2.5. Plan de lanzamiento

Lanzamiento	Orden de las HU a implementar	Duración Total
Iteración 1	HU1, HU2, HU3, HU4	2.9 semanas
Iteración 2	HU5, HU6, HU7, HU8, HU14, HU15	3 semanas
Iteración 3	HU9, HU10, HU11, HU12, HU13	3 semana

2.4.4. Plan de entregas

Esta etapa abarca múltiples rondas de trabajo en el sistema antes de su entrega final. El Plan de Entrega se estructura en iteraciones de duración no superior a tres semanas. En la primera iteración, se explorará la posibilidad de establecer una estructura arquitectónica sólida para el sistema, la cual será la base para el desarrollo continuo a lo largo del proyecto. Este objetivo se alcanza al seleccionar las historias de usuario que impulsen la creación de esta arquitectura, aunque vale la pena destacar que no siempre es factible debido a que es el cliente quien determina qué historias se incluirán en cada iteración, con el propósito de maximizar el valor empresarial. Al concluir la última iteración, el sistema estará preparado para su implementación en producción. En la siguiente tabla se muestran las versiones del producto por historias de usuario.

Tabla 2.6. Plan de entrega

Versión del Producto	Iteración	Módulo/ Funcionalidad	Historias de Usuario (HU)
Versión 1.0	Iteración 1	Usuarios	HU #1: Gestionar Usuario, HU #2: Autenticar usuario
		Personas	HU #3: Gestionar Persona
		Curso Académico	HU #4: Gestionar Curso
		Entrega Final Versión 1.0	Incluye funcionalidades básicas de Usuarios, Personas y Curso Académico

Versión del Producto	Iteración	Módulo/ Funcionalidad	Historias de Usuario (HU)
Versión 1.1	Iteración 2	Facultad	HU #5: Gestionar Facultad
		Carrera	HU #6: Gestionar Carrera
		Grupo	HU #7: Gestionar Grupo
		Distribución	HU #8: Configurar Distribución
		Reportes	HU #14: Reporte provincia, HU #15: Reporte general
		Entrega Final Versión 1.1	Completa la gestión de Cursos, Facultades, Carreras, Grupos, Distribución y Reportes
Versión 1.2	Iteración 3	Distribución	HU #9: Distribución por Facultad, HU #10: Distribución por Grupo
		Reportes	HU #13: Reporte vía de ingreso
		Provincia	HU #11: Gestionar provincia
		Municipio	HU #12: Gestionar municipio
		Entrega Final Versión 1.2	Producto completo con módulos de Distribución, Provincia, Municipio y Reportes

2.5. Fase de diseño

Durante el diseño de software, siempre se busca la solución más simple y eficiente que pueda implementarse en un momento dado del proyecto. Se descarta de inmediato cualquier complejidad innecesaria y código adicional. De acuerdo con la visión de Kent Beck, el diseño ideal para el software en cualquier etapa es aquel que pasa todas las pruebas, evita la duplicación de lógica, expresa claramente la intención de los desarrolladores y minimiza el número de clases y métodos al máximo. Este enfoque en la simplicidad y la eliminación de complicaciones innecesarias contribuye a mantener un código más limpio, fácil de comprender y ágil ante los cambios en los requisitos del proyecto (Beck, s.f.).

2.5.1. Patrones de arquitectura de software

De acuerdo con el *Software Engineering Institute* (SEI), la arquitectura de software hace referencia a "las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos?".

En otras palabras, se refiere a la estructura que debe tener un software, las partes que se van a construir y la forma en que vamos a unirlos y combinarlos para poder trabajar con ellos. Algo así como un plano que servirá de guía para el posterior desarrollo técnico y que facilitará la comunicación entre los diferentes perfiles profesionales que deban trabajar sobre él.

En primer lugar, los desarrolladores del software sabrán cómo se integran todos los componentes. Por ejemplo, la definición de los módulos, las interfaces o la comunicación entre estas. Una segunda función es la de establecer principios y pautas que irán guiando el resto de decisiones sobre desarrollo y diseño. Además de

ser un software de calidad, debe ser también seguro, escalable y fácil de mantener (*Arquitectura de software* 2024).

Patrón arquitectónico **Cliente-Servidor**:

La Arquitectura Cliente-Servidor es un modelo en el que dos componentes principales, el cliente y el servidor, se comunican para completar tareas.

- **Cliente:** Es el dispositivo que solicita servicios o recursos (por ejemplo, una computadora o teléfono que accede a una página web).
- **Servidor:** Es el sistema que recibe, procesa las solicitudes del cliente, y devuelve la respuesta (como un servidor web que envía la página solicitada).

En este modelo, el cliente maneja la interfaz de usuario y el servidor gestiona los datos y la lógica de la aplicación. Este enfoque permite distribuir las tareas, centralizar la información en el servidor y facilita el mantenimiento del sistema (*Client-Server Model* 2019).

En el contexto del sistema se puede ver de la siguiente manera:

El frontend (React): Es la interfaz de usuario que interactúa con el usuario. React se encarga de la lógica de presentación, la renderización de la página y la interacción del usuario.

El backend (Django): Es el servidor que se encarga de la lógica de negocio, el manejo de datos y la API que interactúa con el frontend.

En este caso, Django actúa como un servidor web que proporciona una API RESTful para que React pueda solicitar datos, enviar información y realizar operaciones.

Para un mejor entendimiento del patrón arquitectónico a utilizar en la solución propuesta a continuación se muestra un esquema donde quedan reflejados los aspectos más importantes del patrón arquitectónico seleccionado.

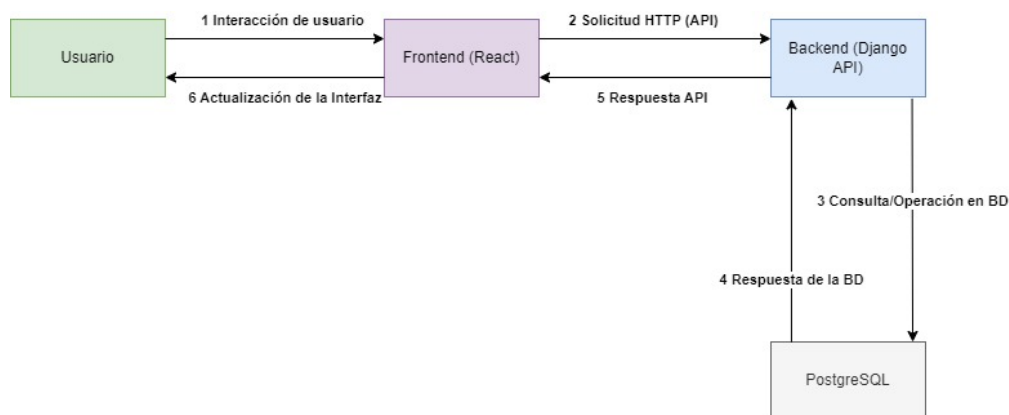


Figura 2.2. Patrón de Arquitectura Cliente-Servidor

Este esquema muestra el flujo:

1. El usuario interactúa con el frontend (React).

2. React envía una solicitud HTTP a la API de Django.
3. Django procesa la solicitud, realiza operaciones en la base de datos.
4. La base de datos devuelve una respuesta al backend.
5. Django devuelve una respuesta.
6. React recibe la respuesta y actualiza la interfaz de usuario.

2.5.2. Patrones de diseño

Los patrones de diseño o design patterns, son una solución general, reutilizable y aplicable a diferentes problemas de diseño de software. Se trata de plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error. Estos patrones proporcionan un enfoque estructurado y reutilizable para resolver situaciones recurrentes en el desarrollo de software. Los patrones de diseño ayudan a los desarrolladores a comunicarse y compartir soluciones eficientes y efectivas que han demostrado ser exitosas en el pasado. Cada patrón de diseño tiene un propósito específico y ofrece una solución a un problema de diseño común. Al utilizar un patrón de diseño, los desarrolladores pueden aprovechar la experiencia acumulada y evitar reinventar la rueda, lo que resulta en un software más eficiente, mantenible y escalable. Es importante tener en cuenta que los patrones de diseño no son algoritmos o código listo para usar, sino más bien pautas y descripciones de soluciones de diseño. Los desarrolladores deben adaptar e implementar los patrones de diseño de acuerdo con los requisitos y características específicas de su proyecto de software. Los patrones de diseño de software se basan en principios y prácticas de diseño que han evolucionado a lo largo del tiempo. A menudo, se describen utilizando terminología y notaciones específicas para facilitar su comprensión y aplicación (Blancarte, 2016).

Patrones GRASP:

Los patrones GRASP (*General Responsibility Assignment Software Patterns*) son un conjunto de patrones de diseño que se enfocan en la asignación de responsabilidades en el diseño de sistemas de software. Estos patrones ayudan a los desarrolladores a tomar decisiones sobre cómo distribuir responsabilidades entre clases y objetos, lo que puede mejorar la claridad, mantenibilidad y cohesión del código (Ortega, 2021).

Patrones utilizados en el sistema a desarrollar:

Experto: El principio de Experto dice que la responsabilidad de una tarea debe recaer en la clase que tenga la información necesaria para realizarla de forma adecuada. En Django, se suele asignar las responsabilidades de forma coherente a cada clase basándose en su conocimiento y habilidades. Las funciones `save()` de las clases persistentes se le asignaron a estas clases porque son ellas las que manejan la información del objeto que van a guardar. Siguiendo el patrón de experto se consigue también una alta cohesión y un bajo acoplamiento. Estas clases son:

- Usuario
- Carrera
- Persona

- Curso Académico
- Facultad
- Grupo
- Distribución
- Provincia
- Municipio

Estas clases tienen como objetivos crear instancias, manipular e interactuar con las clases que representan los modelos del sistema. Controlador: Django implementa el patrón de diseño Controlador al asignar la lógica del controlador a las vistas basadas en clases, donde cada vista puede manejar solicitudes específicas y responder en consecuencia. Por ejemplos en las clases:

- UsuarioView
- CarreraView
- PersonaView
- CursoAcademicoView
- FacultadView
- GrupoView
- DistribuciónView
- ProvinciaView
- MunicipioView

Se puede evidenciar un bajo acoplamiento entre las clases al utilizar los modelos o esquemas de django.db para definir la estructura de datos en lugar de definir estructuras de datos directamente en las rutas. Esto reduce el acoplamiento, ya que se puede reutilizar modelos en múltiples rutas.

En la siguiente ilustración se muestra un ejemplo de clase con modelos de django.db:

```

1  from django.db import models
2  from ..Facultad.models import Facultad
3
4  # Create your models here.
5  class Carrera(models.Model):
6      nombre = models.CharField(max_length=50, null=True, unique=True) # Nombre
7      facultades = models.ManyToManyField(Facultad, related_name='carreras') # Facultad a la que pertenece
8      active = models.BooleanField(default=True) #Atributo para saber si esta activo
9
10     def __str__(self):
11         return self.nombre

```

Figura 2.3. Ejemplo de bajo acoplamiento entre clases

Patrones GoF:

Los patrones GoF (*Gang of Four*) son un grupo de veintitrés patrones de diseño publicados originalmente en un libro fundamental titulado "Patrones de diseño: elementos de software orientado a objetos reutilizable" ("*Design Patterns: Elements of reusable Object-Oriented Software*"); el término "*Gang of Four*" hace

referencia a los cuatro autores. Estos patrones proporcionan una guía sobre cómo estructurar y organizar el código de una manera eficiente y reutilizable, lo que facilita la escalabilidad, el mantenimiento y la claridad del software. Además, son un lenguaje común entre desarrolladores, lo que ayuda en la colaboración y comprensión del código (Duque-Alfonso, 2019).

Patrones utilizados en el sistema a desarrollar:

Patrón Factory Method: La clase `Grupo.objects.get(id=id)` utiliza el patrón Factory Method, en el sentido de que está delegando la creación o recuperación del objeto `Grupo` a un método especializado (`objects.get`). En Django, los managers (`objects`) suelen actuar como fábricas para crear o recuperar instancias del modelo.

Patrón Facade (Fachada): La clase `GrupoView` en conjunto con `APIView` puede ser vista como una implementación del patrón Facade, ya que encapsula la lógica de interacción con la base de datos (a través de `Grupo.objects.get` y `GrupoSerializer`) y proporciona una interfaz simplificada para manejar las solicitudes HTTP. Esto facilita la gestión de solicitudes GET, sin exponer detalles internos del acceso a datos o serialización.

Patrón Template Method (Método Plantilla): `APIView` establece un flujo de trabajo básico para manejar las peticiones HTTP, mientras que el método `get` en `GrupoView` define el comportamiento específico que ocurre cuando se recibe una solicitud GET. Así, la lógica general del manejo de solicitudes está en `APIView`, pero `GrupoView` implementa el método específico, lo cual es característico del patrón Template Method.

```

19 class GrupoView(APIView):
20     def get(self, request, id):
21         try:
22             grupo = Grupo.objects.get(id=id)
23         except Grupo.DoesNotExist:
24             return Response({'Error': 'Grupo no encontrada'}, status=status.HTTP_404_NOT_FOUND)
25
26         serializer = GrupoSerializer(grupo)
27         return Response(serializer.data)
28

```

Figura 2.4. Ejemplo patrones GoF

2.5.3. Tarjetas CRC

Las tarjetas CRC (*Class-Responsibility-Collaboration*) son una herramienta de diseño utilizada en el desarrollo de software para identificar y definir las clases en un sistema, junto con sus responsabilidades y colaboraciones.

Por tanto las tarjetas CRC son una herramienta poderosa en el desarrollo ágil, especialmente en XP, ya que facilitan la identificación y definición de clases de manera colaborativa y visual. Permiten a los equipos de desarrollo adaptarse rápidamente a los cambios, mantener una comunicación efectiva y asegurar que el diseño del software esté alineado con las necesidades del usuario. Al ser un enfoque simple y práctico, se integran bien en la filosofía de XP, que valora la adaptabilidad y la colaboración en el proceso de desarrollo (Tarjetas CRC 2024).

A continuación se muestran las tarjetas CRC definidas:

Tabla 2.7. Tarjeta CRC # 1

Tarjeta CRC	
Clase: CarreraView	
Responsabilidad	Colaboración
• Registrar las carreras • Modificar las carreras • Eliminar las carreras • Buscar las carreras • Mostrar el listado de las carreras	Carrera Facultad Model

Tabla 2.8. Tarjeta CRC # 2

Tarjeta CRC	
Clase: PersonaView	
Responsabilidad	Colaboración
• Importar listado • Modificar las personas • Eliminar las personas • Buscar las personas • Mostrar el listado de las personas • Exportar plantilla vacía • Exportar distribución	Persona Grupo Curso Model

Tabla 2.9. Tarjeta CRC # 3

Tarjeta CRC	
Clase: GrupoView	
Responsabilidad	Colaboración
• Crear Grupos • Modificar los grupos • Eliminar los grupos • Buscar los grupos • Mostrar el listado de los grupos	Grupo Carrera Model

El resto de las tarjetas crc se muestran en el apéndice B de los anexos.

2.6. Algoritmo de Inteligencia Artificial

Se determinó que el uso de un algoritmo de inteligencia artificial era necesario para el proceso de distribución debido a la complejidad inherente del problema, que implica manejar múltiples variables y restricciones, como género, índice académico y provincia de procedencia, mientras se busca una asignación equitativa y eficiente. La naturaleza combinatoria de este tipo de tareas hace que métodos manuales o tradicionales resulten ineficaces o propensos a errores, especialmente cuando se trabaja con grandes volúmenes de datos. El uso de un enfoque basado en inteligencia artificial permite automatizar este proceso, explorar un mayor número de soluciones posibles y garantizar que las decisiones sean consistentes con los criterios definidos. Siguiendo lo antes mencionado se analizaron tres algoritmos de inteligencia artificial como posibles soluciones: el algoritmo genético, las redes neuronales y el algoritmo de colonia de hormigas. Cada uno aborda los problemas de optimización de manera distinta, con fortalezas y debilidades particulares.

El algoritmo genético está inspirado en la teoría de la evolución, utilizando conceptos como selección, cruzamiento y mutación para generar soluciones óptimas. Su flexibilidad y capacidad de manejar múltiples restricciones lo hacen adecuado para problemas combinatorios (*Algoritmos Genéticos* 2024).

Por otro lado, **las redes neuronales artificiales** se basan en estructuras que imitan el cerebro humano, aprendiendo patrones complejos a partir de datos y siendo más útiles en tareas predictivas, aunque menos prácticas para problemas de optimización directa como la distribución (Abeliuk y Gutiérrez, 2021).

Finalmente, **el algoritmo de colonia de hormigas** toma inspiración del comportamiento cooperativo de las hormigas, buscando rutas óptimas mediante la acumulación de feromonas que representan soluciones eficientes (García, 2024).

A continuación, se presenta una tabla comparativa de estos algoritmos en función de las necesidades del sistema:

Tabla 2.10. Comparación entre los algoritmos de inteligencia artificial

Aspectos	Algoritmo Genético	Redes Neuronales	Algoritmo de Colonia de Hormigas
Facilidad de Implementación	Relativamente sencillo, con bibliotecas como PyGAD que facilitan su integración en Python y Django.	Complejo, requiere entrenamiento previo, ajuste de hiperparámetros y mayor tiempo de desarrollo.	Moderadamente complejo, con algoritmos específicos y parámetros que deben ajustarse manualmente.
Tiempo de Procesamiento	Razonable, con resultados rápidos para problemas medianos; depende del tamaño de la población y generaciones.	Lento, especialmente durante el entrenamiento inicial y ajustes de parámetros.	Lento en problemas de gran escala debido a la exploración de muchas rutas posibles.
Capacidad de Optimización	Excelente, explora un espacio amplio de soluciones y encuentra opciones cercanas al óptimo global.	Limitada para problemas combinatorios; necesita enfoques adicionales para manejar restricciones específicas.	Muy buena, encuentra soluciones óptimas iterativamente, pero puede requerir más tiempo de convergencia.
Compatibilidad con Django	Muy compatible, fácil de integrar con la estructura del backend y bases de datos.	Compatible, pero requiere integración de modelos externos o librerías adicionales como TensorFlow o PyTorch.	Compatible, pero requiere más personalización para adaptarse al flujo del sistema.

El algoritmo genético fue seleccionado debido a su flexibilidad para adaptarse a múltiples criterios como género, índice académico y provincia, así como su facilidad de implementación utilizando herramientas disponibles en Python como PyGAD. Además, ofrece un balance ideal entre tiempo de procesamiento y capacidad de optimización, lo que lo hace adecuado para manejar el problema de distribución en el sistema. Aunque el algoritmo de colonia de hormigas también es flexible y poderoso, su implementación es más compleja, y las redes neuronales no son ideales para problemas de optimización combinatoria como este. Por estas razones, el algoritmo genético fue la opción más práctica y eficiente.

Por tanto, se utilizó la biblioteca PyGAD de Python para implementar el algoritmo genético encargado de la distribución de estudiantes. PyGAD, gracias a su flexibilidad y facilidad de integración, permitió configurar parámetros como el número de generaciones, tamaño de la población y operadores genéticos, adaptándolos a las necesidades específicas del problema (Fawzy Gad, 2021).

2.6.1. Algoritmo Genético

Componentes principales de un algoritmo genético:

1. **Población:** Un conjunto de posibles soluciones al problema, llamadas individuos o cromosomas.
2. **Función de fitness:** Evalúa qué tan buena es cada solución en la población. Las mejores soluciones tienen más probabilidades de reproducirse.
3. **Selección:** Se seleccionan los individuos más aptos de la población para reproducirse. La selección puede ser aleatoria, pero las soluciones con mejor fitness tienen una mayor probabilidad de ser elegidas.
4. **Cruzamiento (crossover):** Dos individuos combinan sus características para producir uno o más descendientes. Esto permite que la nueva generación herede características de las soluciones más exitosas.
5. **Mutación:** Se aplican pequeñas alteraciones aleatorias en los descendientes para mantener la diversidad genética y evitar la convergencia prematura en una solución subóptima.

Cómo funciona un algoritmo genético:

1. **Inicialización:** Se crea una población inicial de soluciones aleatorias.
2. **Evaluación:** Cada individuo de la población es evaluado mediante la función de fitness.
3. **Selección, cruzamiento y mutación:** Los individuos más aptos se seleccionan para cruzarse y generar nuevos individuos. Se aplican mutaciones para agregar variabilidad.
4. **Reemplazo:** La nueva generación reemplaza a la anterior.
5. **Iteración:** El proceso se repite hasta que se cumpla un criterio de parada, como alcanzar un número de generaciones o una solución con un nivel de fitness satisfactorio.

Dada la complejidad inherente en la distribución y asignación de estudiantes en el sistema, el algoritmo genético se presenta como una solución adecuada debido a su capacidad de explorar un espacio de búsqueda amplio y su flexibilidad para ajustarse a los cambios en las condiciones de entrada. Esto permite optimizar la distribución de manera que cumpla con criterios específicos, como balancear el número de estudiantes entre facultades y grupos.

Descripción del Funcionamiento del Algoritmo Genético en la Distribución:

En el contexto de este sistema, cada individuo de la población representa una configuración específica de asignación de estudiantes a distintas facultades y grupos. La función de fitness mide el grado de balanceo entre facultades y la adecuación a las preferencias. Los individuos más óptimos se seleccionan para el cruzamiento, generando nuevas distribuciones y aplicando mutaciones aleatorias para evitar la convergencia prematura en una solución subóptima.

- **Población inicial:** Cada individuo en la población representa una posible configuración de distribución.

- **Función de fitness:** Evalúa qué tan bien cada distribución cumple con los criterios establecidos.
- **Selección, cruzamiento y mutación:** Explica cómo se eligen y combinan las mejores configuraciones para generar nuevas distribuciones y cómo se introducen variaciones para mejorar la búsqueda de la solución óptima.

Codificación empleada en la distribución por Facultad: El algoritmo utiliza una codificación entera para representar las soluciones.

- Cada gen en una solución es un índice entero que representa una facultad a la que un estudiante ha sido asignado.
- Por ejemplo, una solución podría ser un vector como [0, 2, 1, 3], donde cada número representa la facultad asignada a cada estudiante en orden.

Función de Fitness Empleada:

La función de fitness en la distribución por facultad evalúa la calidad de cada solución basada en tres criterios principales:

- **Balance de Género:** Calcula qué tan equilibrada está la distribución de géneros dentro de cada facultad.

$$\text{Balance género} = 1 - \text{Diferencia proporcional entre géneros}$$

- **Homogeneidad del Índice Académico:** Penaliza la desviación estándar de los promedios académicos de los estudiantes asignados a una misma facultad.

$$\text{Homogeneidad índice} = 1 - \text{Desviación estándar}$$

- **Diversidad por Provincia:** Evalúa qué tan diversas son las provincias de los estudiantes dentro de cada facultad utilizando el Índice de Simpson.

$$\text{Diversidad provincia} = 1 - \sum \left(\frac{\text{Estudiantes de provincia}}{\text{Total estudiantes}} \right)^2$$

Cada criterio está ponderado por factores definidos en el diccionario pesos:

- **Peso género:** 1.0
- **Peso índice académico:** 0.8
- **Peso diversidad por provincia:** 0.5

La función de fitness totaliza estas métricas para calcular la puntuación final de cada solución.

Operadores Utilizados:

- **Selección:** Aunque no se especifica explícitamente en el código, PyGAD usa por defecto la selección de torneos para elegir a los padres. Esto se basa en comparar un subconjunto aleatorio de soluciones y elegir el mejor.
- **Cruzamiento:** PyGAD emplea un cruzamiento de un punto, donde los genes se mezclan en un punto aleatorio entre dos padres.
- **Mutación:** Se utiliza mutación por intercambio, donde se cambia aleatoriamente un gen por otro valor dentro del espacio de genes permitido (gene space).

Conclusiones del capítulo

En este segundo capítulo, se ha trazado el camino para desarrollar la solución al problema científico. A continuación, se destacan las conclusiones más relevantes:

- La elección de la metodología de desarrollo de software facilitó la estructuración y establecimiento de procedimientos claros para implementar el sistema, permitiendo además crear los artefactos esenciales para cada fase propuesta. Asimismo, contribuyó a generar los elementos necesarios para las etapas de planificación y diseño del proyecto.
- Las historias de usuario fueron clave para definir los requisitos del sistema, ya que ofrecen una descripción clara y accesible de las funciones y características que los clientes esperan ver reflejadas.
- Tras recolectar los requisitos mediante las historias de usuario, que permitieron estimar el tiempo y esfuerzo requeridos, se definieron las iteraciones de trabajo en función del cronograma de implementación.
- El uso de una arquitectura cliente-servidor permitió una clara separación entre la interfaz de usuario y la lógica del sistema, facilitando la escalabilidad, el desarrollo modular y la independencia entre el cliente y el servidor. Esta arquitectura asegura que las funcionalidades puedan evolucionar de manera autónoma en ambos lados, optimizando tanto el rendimiento como la experiencia del usuario.
- El uso del algoritmo genético proporciona una gran flexibilidad, permitiendo ajustes constantes en función de las condiciones de entrada o los criterios de asignación. Esto asegura una distribución optimizada y equitativa de estudiantes.

Implementación y pruebas

En este capítulo, se presenta una descripción detallada de las tareas de ingeniería que surgieron en el curso de la implementación de cada una de las Historias de Usuario. Además, se exponen tanto las pruebas de aceptación como las pruebas unitarias que se planificaron meticulosamente para el sistema. El propósito fundamental de estas acciones es garantizar la entrega de un producto final que haya sido sometido a rigurosas pruebas y esté plenamente funcional para el cliente. En esta sección, se abordarán las tareas y desafíos técnicos que emergieron en la ejecución de cada Historia de Usuario. Asimismo, se explorará en detalle el proceso de planificación y ejecución de pruebas, abordando tanto las pruebas de aceptación, que aseguran que el sistema cumpla con las expectativas del cliente, como las pruebas unitarias, que verifican el funcionamiento adecuado de componentes individuales. Este enfoque de ingeniería integral garantiza que el producto final entregado al cliente esté respaldado por un sólido proceso de desarrollo y pruebas, y cumpla con los más altos estándares de calidad y funcionalidad.

3.1. Fase de Implementación

La fase de codificación o implementación en la metodología XP es una etapa fundamental en el proceso de desarrollo de software, donde se lleva a cabo la implementación del software a partir de las historias de usuario identificadas en la fase de planificación y el diseño realizado en la fase anterior. Se realiza bajo estándares que mantienen el código consistente y facilitan su comprensión y escalabilidad, y se lleva a cabo en parejas, utilizando pruebas unitarias continuas y pruebas de aceptación para asegurar la calidad del software.

3.1.1. Tareas ingenieriles.

Las tareas de ingeniería son utilizadas para identificar, controlar y definir cada una de las actividades que dan cumplimiento a las historias de usuarios con el objetivo de facilitar la construcción del sistema. Estas tareas son fundamentales para el desarrollo exitoso de un sistema, debido a que permiten una gestión eficiente de

los recursos y una mayor adaptabilidad a los cambios (Pressman y Maxim, 2020).

A continuación se muestran las Tareas Ingenieriles definidas:

Tabla 3.1. Tarea de ingeniería # 1

Tarea	
Número de tarea: 1	Número de Historia de usuario: 1
Nombre de la tarea: Gestionar Usuario	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión de la información de las personas.	

Tabla 3.2. Tarea de ingeniería # 2

Tarea	
Número de tarea: 2	Número de Historia de usuario: 2
Nombre de la tarea: Autenticar Usuario	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar el método e interfaz necesaria para el autenticar usuario	

Tabla 3.3. Tarea de ingeniería # 3

Tarea	
Número de tarea: 3	Número de Historia de usuario: 1
Nombre de la tarea: Gestionar Persona	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Gabriela de la C. Rodríguez Rey	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión de la información de las personas.	

El resto de las tareas ingenieriles se muestran en el apéndice C de los anexos.

Después de definir las tareas de ingeniería necesarias para la implementación de todas las funcionalidades del sistema, surge la necesidad de comprobar su correcto funcionamiento y la corrección de errores en caso de presentarse, a continuación, se darán más detalles del proceso de pruebas.

3.2. Fase de Pruebas

En la metodología de Extreme Programming (XP), las pruebas juegan un papel esencial y son fundamentales para garantizar la calidad del software entregado. XP promueve la integración continua, lo que significa que

las pruebas se realizan de manera constante durante todo el proceso de desarrollo. En particular, se hace énfasis en las pruebas unitarias y las pruebas de aceptación automatizadas, las cuales permiten identificar errores tempranos y asegurar que cada funcionalidad cumpla con los requisitos del cliente. Las pruebas unitarias son realizadas frecuentemente por los mismos desarrolladores mientras codifican, mientras que las pruebas de aceptación se enfocan en validar que el software cumpla con los criterios establecidos por el cliente (Letelier y PenadÃ©s, 2021).

3.2.1. Pruebas de aceptación

Las pruebas de aceptación en metodologías ágiles, como en el marco de trabajo de XP, se utilizan para validar que una característica o historia de usuario cumpla con los requisitos y expectativas del cliente o usuario final. Estas pruebas están basadas en criterios predefinidos, que especifican las condiciones que deben cumplirse para que una funcionalidad sea considerada completaz pueda ser entregada (*8 Reasons Why You Should Use Acceptance Criteria in Agile Testing* 2024).

El objetivo principal de las pruebas de aceptación es asegurar que el producto final cumpla con los criterios de calidad y funcionalidad acordados entre los desarrolladores y los stakeholders, garantizando que lo entregado al cliente sea exactamente lo que se espera (CCF2, 2024).

A continuación, se describen los casos de pruebas de aceptación definidos:

Tabla 3.4. Prueba de aceptación # 1

Caso de prueba de aceptación	
Código: HU1_P1	Historia de usuario: 1
Nombre: Listar Usuario	
Descripción: Prueba la funcionalidad de listar los usuarios.	
Condiciones de ejecución: El administrador debe estar autenticado.Debe haber al menos un usuario creado.	
Pasos de ejecución: El administrador selecciona en el menú lateral la sección Seguridad y después selecciona la subsección Usuario.	
Resultados esperados: El sistema carga una página con el listado de usuarios.	

Tabla 3.5. Prueba de aceptación # 2

Caso de prueba de aceptación	
Código: HU1_P2	Historia de usuario: 1
Nombre: Registrar Usuario	
Descripción: Prueba la funcionalidad de registrar un usuario.	

Continúa en la próxima página

Tabla 3.5. Continuación de la página anterior

Condiciones de ejecución: El administrador debe estar autenticado.
Pasos de ejecución: El administrador selecciona el botón de Añadir, el sistema carga un formulario, el administrador introduce los datos necesarios y verifica que sean correctos, el administrador presiona el botón Aceptar.
Resultados esperados: El sistema registra un nuevo usuario si los datos fueron correctos, en caso contrario muestra un mensaje de error.

Tabla 3.6. Prueba de aceptación # 3

Caso de prueba de aceptación	
Código: HU1_P3	Historia de usuario: 1
Nombre: Editar Usuario	
Descripción: Prueba la funcionalidad de editar un usuario.	
Condiciones de ejecución: El administrador debe estar autenticado, debe haber al menos un usuario creado.	
Pasos de ejecución: El administrador selecciona el botón de Editar, el sistema carga un formulario con los datos del usuario, el administrador modifica los datos necesarios y verifica que sean correctos, el administrador presiona el botón Aceptar.	
Resultados esperados: El sistema modifica un usuario si los datos fueron correctos, en caso contrario muestra un mensaje de error.	

Tabla 3.7. Prueba de aceptación # 4

Caso de prueba de aceptación	
Código: HU1_P4	Historia de usuario: 1
Nombre: Eliminar Usuario	
Descripción: Prueba la funcionalidad de eliminar un usuario.	
Condiciones de ejecución: El administrador debe estar autenticado, debe haber al menos un usuario creado.	
Pasos de ejecución: El administrador selecciona el botón de Eliminar, el sistema muestra un cuadro de confirmación, el administrador presiona el botón Aceptar.	
Resultados esperados: El sistema elimina un usuario.	

El resto de las pruebas de aceptación se muestran en el apéndice E de los anexos.

Análisis de las pruebas de aceptación

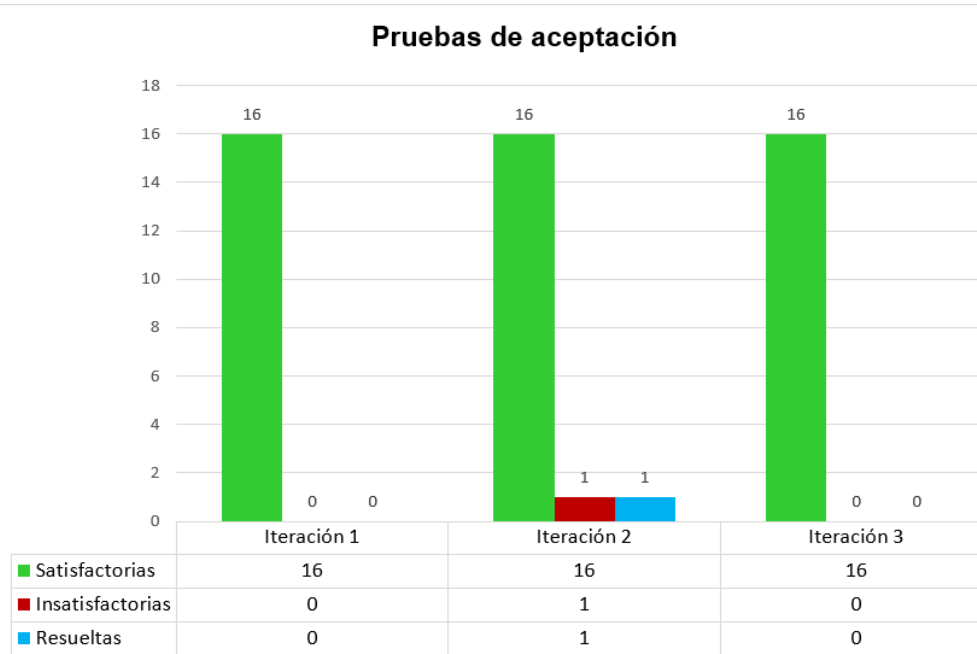


Figura 3.1. Pruebas de Aceptación

En la gráfica anterior se muestra como en la iteración 1 y 3 todos los casos de pruebas arrojaron resultados satisfactorios, no siendo así la segunda iteración en la que, de un total de 17 casos de pruebas, arrojó resultado insatisfactorio uno de los casos propuestos, el caso de prueba 28, esta falla fue resuelta posteriormente en la misma iteración correspondiente debido a que no presentaban gran complejidad de resolución. El error se detectó cuando al intentar guardar la distribución, no se actualizaban los datos de la tabla provocando la visualización incorrecta de los datos, dicha detección permitió corregir el fallo detectado para lograr un 100 por ciento de los casos de pruebas con resultados satisfactorios.

3.2.2. Pruebas unitarias

Las pruebas unitarias son un proceso en el que se prueba la unidad funcional de código más pequeña. Estas pruebas son una parte importante de la creación de software, debido a que ayudan a garantizar la calidad del código y a detectar errores temprano en el proceso de desarrollo. Las pruebas unitarias deben activarse en diferentes eventos dentro del desarrollo de software, como antes de enviar los cambios a una ramificación mediante un software de control de versiones, o antes de implementar una actualización de software (Letelier y PenadÃ©s, 2021). Para estas se usó una de las librerías para las pruebas unitarias en Django, la librería para pruebas unitarias APITestCase.

A continuación, se muestran los resultados de la aplicación de estas pruebas:

```

System check identified no issues (0 silenced).
test_crear_usuario (apps.Usuario.tests.UsuarioTests.test_crear_usuario) ... ok
test_crear_usuario_datos_invalidos (apps.Usuario.tests.UsuarioTests.test_crear_usuario_datos_invalidos) ... ok
test_editar_usuario (apps.Usuario.tests.UsuarioTests.test_editar_usuario) ... ok
test_editar_usuario_no_existente (apps.Usuario.tests.UsuarioTests.test_editar_usuario_no_existente) ... ok
test_eliminar_usuario (apps.Usuario.tests.UsuarioTests.test_eliminar_usuario) ... ok
test_eliminar_usuario_no_existente (apps.Usuario.tests.UsuarioTests.test_eliminar_usuario_no_existente) ... ok
test_listar_usuarios (apps.Usuario.tests.UsuarioTests.test_listar_usuarios) ... ok
test_obtener_usuario (apps.Usuario.tests.UsuarioTests.test_obtener_usuario) ... ok
test_obtener_usuario_no_existente (apps.Usuario.tests.UsuarioTests.test_obtener_usuario_no_existente) ... ok

-----
Ran 9 tests in 0.641s

OK

```

Figura 3.2. Pruebas unitarias realizadas a la ruta usuario

```

System check identified no issues (0 silenced).
test_crear_persona (apps.Persona.tests.PersonaTests.test_crear_persona) ... ok
test_editar_persona (apps.Persona.tests.PersonaTests.test_editar_persona) ... ok
test_eliminar_persona (apps.Persona.tests.PersonaTests.test_eliminar_persona) ... ok
test_generar_plantilla_personas (apps.Persona.tests.PersonaTests.test_generar_plantilla_personas) ... ok
test_listar_personas (apps.Persona.tests.PersonaTests.test_listar_personas) ... ok
test_obtener_persona (apps.Persona.tests.PersonaTests.test_obtener_persona) ... ok

-----
Ran 6 tests in 0.045s

OK

```

Figura 3.3. Pruebas unitarias realizadas a la ruta persona

El resto de los resultados de las pruebas unitarias se muestran en el apéndice *D* de los anexos.

Análisis de las pruebas de unitarias

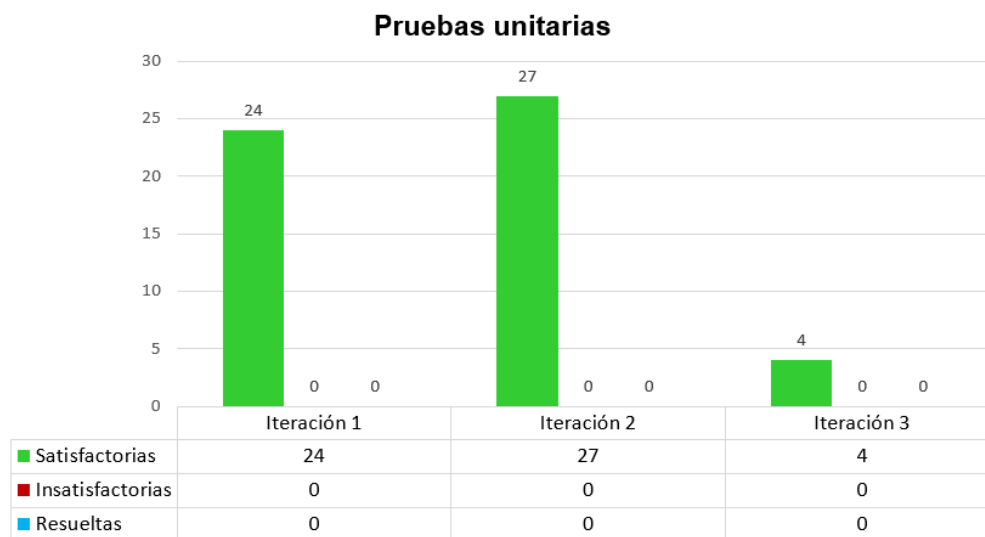


Figura 3.4. Pruebas Unitarias

En la gráfica anterior se muestra como en la iteración 1, 2 y 3 todas las pruebas unitarias arrojaron resultados satisfactorios, lográndose un 100 por ciento de las pruebas unitarias con resultados satisfactorios

Conclusiones del capítulo

Luego del desarrollo del presente capítulo sobre el sistema de desarrollado, se arribó a las siguientes conclusiones:

- El proceso de implementación del sistema fue llevado a cabo por medio de las tareas de ingeniería que permitió el desarrollo del 100 por ciento de las funcionalidades del sistema.
- Se llevaron a cabo las pruebas de aceptación, dando como resultados un 100 por ciento de casos de pruebas satisfactorios para la iteración 1 y 3, y un 93 por ciento de resultados favorables para la iteración 2.

Luego del desarrollo de la presente investigación sobre el sistema de gestión para el ingreso y distribución de los nuevos estudiantes de la Universidad de las Ciencias Informáticas (UCI), se arribó a las siguientes conclusiones:

- La creación de los artefactos y entregables propuestos por la metodología de desarrollo seleccionada permitieron una mayor comprensión y organización del sistema desarrollado.
- La utilización de Django en el backend y React en el frontend permitió la implementación de una aplicación web con características de seguridad, mantenimiento eficiente y flexibilidad.
- La adopción de un sistema de gestión basado en una aplicación web proporciona un marco para la distribución de estudiantes y la interacción con los administradores. El sistema permite gestionar la asignación de estudiantes a facultades y grupos, facilitando la toma de decisiones y la generación de reportes.
- La implementación de un algoritmo genético en el proceso de distribución permitió diseñar una solución basada en técnicas de optimización, considerando variables como género, índice académico y provincia de procedencia. Esto ofrece una herramienta para abordar escenarios complejos de asignación con criterios definidos.

Recomendaciones

Debido al continuo proceso de mejora y evolución que caracteriza a cualquier sistema de software, se sugieren las siguientes acciones:

- La incorporación de una sección para el traslado de personas entre grupos y/o facultades.
- La incorporación de una sección para la gestión de funcionalidades y roles del sistema.

Referencias bibliográficas

- 8 Reasons Why You Should Use Acceptance Criteria in Agile Testing*, 2024 [online] [visitado 2024-11-13]. Disponible desde: <https://agile33.com/8-reasons-why-you-should-use-acceptance-criteria-in-agile-testing/> (vid. pág. 44).
- ABELIUK, Andrés y GUTIÉRREZ, Claudio, 2021. Historia y evolución de la inteligencia artificial. *Revista Bits de Ciencia*. N.º 21, págs. 14-21 (vid. pág. 37).
- Acerca de EduAdmin*, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://www.eduadmin.com/about.htm> (vid. pág. 5).
- Algoritmos GenÁticos*, 2024 [online] [visitado 2024-10-30]. Disponible desde: https://www.cs.us.es/~fsancho/Blog/posts/Algoritmos_Geneticos.md.html (vid. pág. 37).
- ALICE, 2021. *Extreme Programming (XP): what it is and how it works* [online] [visitado 2024-08-17]. Disponible desde: <https://www.ryadel.com/en/extreme-programming-xp-what-is-how-works/> (vid. pág. 8).
- Angular* [online] [visitado 2024-08-02]. Disponible desde: <https://angular.dev/> (vid. pág. 18).
- Arquitectura de software, 2024. *Arquitectura de software: 5 patrones principales* [online] [visitado 2024-10-27]. Disponible desde: <https://www.inesdi.com/blog/arquitectura-de-software-5-patrones-principales/> (vid. pág. 30).
- BECK, Kent. *Extreme Programming Explained: Embrace Change* (vid. pág. 29).
- BLANCARTE, Oscar, 2016. Introducción a los Patrones de Diseño. *México, México DF* (vid. pág. 32).
- CCF2, 2024. *Acceptance Standards In Agile Testing* [online] [visitado 2024-11-13]. Disponible desde: <https://consultants-forum.com/2024/09/04/acceptance-standards-in-agile-testing/> (vid. pág. 44).
- Client-Server Model*, 2019 [online] [visitado 2024-10-30]. Disponible desde: <https://www.geeksforgeeks.org/client-server-model/>. Section: Web Technologies (vid. pág. 30).
- CSS, 2024. *CSS: Hojas de estilo en cascada | MDN* [online] [visitado 2024-08-02]. Disponible desde: <https://developer.mozilla.org/en-US/docs/Web/CSS> (vid. pág. 14).
- Django*, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://www.djangoproject.com/> (vid. pág. 15).

- DUQUE-ALFONSO, Juan Pablo, 2019. Método de predicción automático para evaluar los atributos de calidad en patrones de diseño basados en programación orientada a objetos durante la etapa de desarrollo de aplicaciones de software (vid. pág. 34).
- Encuesta para desarrolladores de Stack Overflow 2024* [online] [visitado 2024-08-02]. Disponible desde: <https://survey.stackoverflow.co/2024/> (vid. pág. 18).
- FastAPI - FastAPI*, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://fastapi.tiangolo.com/> (vid. pág. 15).
- FAWZY GAD, Ahmed, 2021. PyGAD: An intuitive genetic algorithm Python library. *arXiv e-prints*, págs. arXiv-2106 (vid. pág. 38).
- Flask web framework*, 2024 [online] [visitado 2024-08-02]. Disponible desde: [https://en.wikipedia.org/w/index.php?title=Flask_\(web_framework\)&oldid=1222790474](https://en.wikipedia.org/w/index.php?title=Flask_(web_framework)&oldid=1222790474). Page Version ID: 1222790474 (vid. pág. 16).
- GARCÍA, Alicia Anierte, 2024. ALGORITMO DE COLONIA DE HORMIGAS APLICADO AL PROBLEMA DEL VIAJANTE (vid. pág. 37).
- Guía de Scrum | Guías de Scrum*, 2024 [online] [visitado 2024-08-17]. Disponible desde: <https://scrumguides.org/scrum-guide.html> (vid. pág. 9).
- Higher Education Software Solutions - Jenzabar*, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://jenzabar.com> (vid. pág. 5).
- HTML, 2024. *HTML: lenguaje de marcado de hipertexto | MDN* [online] [visitado 2024-08-02]. Disponible desde: <https://developer.mozilla.org/en-US/docs/Web/HTML> (vid. pág. 14).
- Java (programming language)*, 2024 [online] [visitado 2024-08-02]. Disponible desde: [https://en.wikipedia.org/w/index.php?title=Java_\(programming_language\)&oldid=1237403459](https://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=1237403459). Page Version ID: 1237403459 (vid. pág. 12).
- JavaScript | MDN*, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (vid. págs. 12, 14).
- LANDRI, Paolo, 2020. *Educational Leadership, Management, and Administration through Actor-Network Theory* [online]. 1.ª ed. Routledge [visitado 2024-08-02]. ISBN 978-0-429-47049-3. Disponible desde DOI: 10.4324/9780429470493 (vid. págs. 4, 5).
- LETELIER, Patricio y PENADÁS, M^a Carmen, 2021. M^a metodologías Ágiles para el desarrollo de software: eXtreme Programming (XP). *Ingeniería del agua* [online]. Vol. 18, n.º 1, págs. ix [visitado 2024-10-26]. ISSN 1886-4996, 1134-2196. ISSN 1886-4996, 1134-2196. Disponible desde DOI: 10.4995/ia.2014.3293 (vid. págs. 25-28, 44, 47).
- MOLINA HERNÁNDEZ, Yenisel; GRANDA DIHIGO, Ailec y VELÁZQUEZ CINTRA, Alionuska, 2019. Los requisitos no funcionales de software. Una estrategia para su desarrollo en el Centro de Informática Médica. *Revista Cubana de Ciencias Informáticas*. Vol. 13, n.º 2, págs. 77-90 (vid. pág. 25).

ORTEGA, Gilberto Andrés Vargas, 2021. Lineamientos para el diseño de aplicaciones web soportados en patrones GRASP. *Ciencia e Ingeniería: Revista de investigación interdisciplinar en biodiversidad y desarrollo sostenible, ciencia, tecnología e innovación y procesos productivos industriales*. Vol. 8, n.º 2, págs. 4 (vid. pág. 32).

pgAdmin User Management in Server Mode, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://enterprisedb.com/blog/pgadmin-user-management> (vid. pág. 21).

pgAdmin User Management in Server Mode | EDB, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://www.enterprisedb.com/blog/pgadmin-user-management> (vid. pág. 21).

PowerSchool K-12 Software & Cloud-Based Solutions, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://www.powerschool.com/> (vid. pág. 5).

PRESSMAN, Roger S. y MAXIM, Bruce R., 2020. *Software engineering: a practitioner's approach*. Ninth edition. New York, NY: McGraw-Hill Education. ISBN 978-1-260-54800-6 978-1-259-87297-6 (vid. págs. 7, 11, 43).

React, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://react.dev/> (vid. pág. 18).

SILBERSCHATZ, Abraham; KORTH, Henry F. y SUDARSHAN, S., 2020. *Database system concepts*. Seventh edition. New York, NY: McGraw-Hill. ISBN 978-0-07-802215-9 978-1-260-51504-6 (vid. pág. 20).

Tarjetas CRC, 2024 [online] [visitado 2024-10-27]. Disponible desde: <http://www.extremeprogramming.org/rules/crccards.html> (vid. pág. 36).

The Agile Unified Process (AUP) Home Page, 2024 [online] [visitado 2024-08-17]. Disponible desde: <https://people.computing.clemson.edu/~johnmc/courses/cpsc372/resources/The%20Agile%20Unified%20Process%20%28AUP%29%20Home%20Page.htm> (vid. pág. 9).

Visual Paradigm Reviews, Pricing and Features, 2024 [online] [visitado 2024-08-17]. Disponible desde: <https://www3.technologyevaluation.com/solutions/53729/visual-paradigm> (vid. pág. 11).

Visual Studio Code - Code Editing. Redefined, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://code.visualstudio.com/> (vid. pág. 22).

Vue.js, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://vuejs.org/> (vid. pág. 18).

Welcome to Python.org, 2024 [online] [visitado 2024-08-02]. Disponible desde: <https://www.python.org/doc/> (vid. pág. 12).

Apéndice

Historias de Usuario

Tabla A.1. Historia de usuario # 4

Historia de usuario	
Número: 4	Nombre: Gestionar Curso académico
Usuario: Administrador, Secretaria	
Prioridad en negocio: Alta	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 1
Programador responsable: Gabriela de la C. Rodríguez Rey	
Descripción: El sistema debe permitir al usuario añadir un curso a partir del campo nombre. Además debe permitir modificar, buscar y ver el listado de todos los cursos.	
Observaciones: El usuario debe estar previamente autenticado.	

Tabla A.2. Historia de usuario # 5

Historia de usuario	
Número: 5	Nombre: Gestionar Facultad
Usuario: Administrador, Secretaria	
Prioridad en negocio: Alta	Riesgo en desarrollo: N/A
Puntos estimados: 2	Iteración asignada: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al usuario añadir una facultad a partir de los campos nombre, nomenclador y siglas. Además debe permitir modificar, buscar y ver el listado de todas las facultades.	
Observaciones: El usuario debe estar previamente autenticado.	

Tabla A.3. Historia de usuario # 6

Historia de usuario	
Número: 6	Nombre: Gestionar Carrera

Continúa en la próxima página

Tabla A.3. Continuación de la página anterior

Usuario: Administrador, Secretaria	
Prioridad en negocio: Media	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 2
Programador responsable: Gabriela de la C. Rodríguez Rey	
Descripción: El sistema debe permitir al usuario añadir una carrera a partir de los campos nombre, nomenclador y siglas. Además debe permitir modificar, buscar y ver el listado de todas las carreras.	
Observaciones: El usuario debe estar previamente autenticado. Deben estar creadas las facultades.	

Tabla A.4. Historia de usuario # 7

Historia de usuario	
Número: 7	Nombre: Gestionar Grupo
Usuario: Administrador, Gerente	
Prioridad en negocio: Media	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 2
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al usuario crear los grupos. Además debe permitir editar, buscar y ver el listado de todos los grupos.	
Observaciones: El usuario debe estar previamente autenticado. Debe estar configurada una distribución.	

Tabla A.5. Historia de usuario # 8

Historia de usuario	
Número: 8	Nombre: Configurar Distribución
Usuario: Administrador, Secretaria	
Prioridad en negocio: Media	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 2
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al usuario crear una configuración de la distribución, incertando valores en una tabla con la cantidad de estudiantes que habrá en cada facultad, por cada carrera y tipo de curso.	
Observaciones: El usuario debe estar previamente autenticado.	

Tabla A.6. Historia de usuario # 9

Historia de usuario	
Número: 9	Nombre: Distribuir por Facultad
Usuario: Administrador, Secretaria	
Prioridad en negocio: Alta	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 3

Continúa en la próxima página

Tabla A.6. Continuación de la página anterior

Programador responsable: Nelson Castroman Aguiar
Descripción: El sistema debe permitir al usuario distribuir las personas por facultad. Además debe permitir seleccionar la carrera por la que comenzará a distribuir por facultades.
Observaciones: El usuario debe estar previamente autenticado. Deben estar creadas las facultades y las carreras.

Tabla A.7. Historia de usuario # 10

Historia de usuario	
Número: 10	Nombre: Distribuir por Grupo
Usuario: Administrador, Secretaria	
Prioridad en negocio: Alta	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 3
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al usuario distribuir las personas por grupo. Además debe permitir seleccionar la facultad por la que comenzará a distribuir por grupos.	
Observaciones: El usuario debe estar previamente autenticado. Deben estar creadas las facultades y los grupos, además de haberse realizado la distribución por facultad.	

Tabla A.8. Historia de usuario # 11

Historia de usuario	
Número: 11	Nombre: Gestionar Provincia
Usuario: Administrador, Secretaria	
Prioridad en negocio: Baja	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 3
Programador responsable: Gabriela de la C. Rodríguez Rey	
Descripción: El sistema debe permitir al usuario añadir una provincia a partir de los campos nombre y código y abreviatura. Además debe permitir editar, eliminar, buscar y ver el listado de todas las provincias.	
Observaciones: El usuario debe estar previamente autenticado.	

Tabla A.9. Historia de usuario # 12

Historia de usuario	
Número: 12	Nombre: Gestionar Municipio
Usuario: Administrador, Secretaria	
Prioridad en negocio: Baja	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 3
Programador responsable: Gabriela de la C. Rodríguez Rey	

Continúa en la próxima página

Tabla A.9. Continuación de la página anterior

Descripción: El sistema debe permitir al usuario añadir un municipio a partir de los campos nombre y código y provincia a la que pertenece. Además debe permitir editar, eliminar, buscar y ver el listado de todos los municipios.
Observaciones: El usuario debe estar previamente autenticado. Deben estar creadas las provincias.

Tabla A.10. Historia de usuario # 13

Historia de usuario	
Número: 13	Nombre: Reporte vía de ingreso
Usuario: Administrador, Secretaria	
Prioridad en negocio: Baja	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 3
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al usuario visualizar un listado de los grupos con la cantidad total de personas que tiene y la cantidad que entraron por cada vía de ingreso.	
Observaciones: El usuario debe estar previamente autenticado.	

Tabla A.11. Historia de usuario # 14

Historia de usuario	
Número: 14	Nombre: Reporte provincia
Usuario: Administrador, Secretaria	
Prioridad en negocio: Baja	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 2
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al usuario visualizar un listado de los grupos con la cantidad total de personas que tiene y la cantidad que hay de cada provincia.	
Observaciones: El usuario debe estar previamente autenticado.	

Tabla A.12. Historia de usuario # 15

Historia de usuario	
Número: 15	Nombre: Reporte general
Usuario: Administrador	
Prioridad en negocio: Baja	Riesgo en desarrollo: N/A
Puntos estimados: 1	Iteración asignada: 2
Programador responsable: Nelson Castroman Aguiar	
Descripción: El sistema debe permitir al usuario visualizar un listado de los grupos con la cantidad total de personas que tiene, la cantidad total del sexo femenino, la cantidad total del sexo masculino y la cantidad que hay de cada provincia.	

Continúa en la próxima página

Tabla A.12. Continuación de la página anterior

Observaciones: El usuario debe estar previamente autenticado.
--

Tarjetas CRC

Tabla B.1. Tarjeta CRC # 4

Tarjeta CRC	
Clase: FacultadView	
Responsabilidad	Colaboración
• Registrar las facultades • Modificar las facultades • Buscar las facultades • Mostrar el listado de las facultades	Facultad Model

Tabla B.2. Tarjeta CRC # 5

Tarjeta CRC	
Clase: CursoAcademicoView	
Responsabilidad	Colaboración
• Registrar los cursos • Modificar los cursos • Buscar los cursos • Mostrar el listado de los cursos	cursoacademico model

Tabla B.3. Tarjeta CRC # 6

Tarjeta CRC	
Clase: DistribucionView	
Responsabilidad	Colaboración

Continúa en la próxima página

Tabla B.3. Continuación de la página anterior

• Registrar la distribucion • Modificar la distribucion • Distribuir por facultad • Distribuir por grupo • Mostrar el listado de la distribucion activa	carrera persona facultad grupo distribución model
---	--

Tabla B.4. Tarjeta CRC # 7

Tarjeta CRC	
Clase: UsuarioView	
Responsabilidad	Colaboración
• Registrar los usuarios • Modificar los usuarios • Eliminar los usuarios • Buscar los usuarios • Mostrar el listado de los usuarios	usuario model

Tabla B.5. Tarjeta CRC # 8

Tarjeta CRC	
Clase: ProvinciaView	
Responsabilidad	Colaboración
• Registrar las provincias • Modificar las provincias • Eliminar las provincias • Buscar las provincias • Mostrar el listado de las provincias	persona provincia municipio model

Tabla B.6. Tarjeta CRC # 9

Tarjeta CRC	
Clase: MunicipioView	
Responsabilidad	Colaboración

Continúa en la próxima página

Tabla B.6. Continuación de la página anterior

• Registrar los municipios • Modificar los municipios • Eliminar los municipios • Buscar los municipios • Mostrar el listado de los municipios	persona provincia municipio model
--	---

Tareas de Ingeniería

Tabla C.1. Tarea de ingeniería # 4

Tarea	
Número de tarea: 4	Número de Historia de usuario: 4
Nombre de la tarea: Gestionar Curso Académico	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Gabriela de la C. Rodríguez Rey	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión de la información de los cursos.	

Tabla C.2. Tarea de ingeniería # 5

Tarea	
Número de tarea: 5	Número de Historia de usuario: 5
Nombre de la tarea: Gestionar Facultad	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión de la información de las facultades.	

Tabla C.3. Tarea de ingeniería # 6

Tarea	
Número de tarea: 6	Número de Historia de usuario: 6
Nombre de la tarea: Gestionar Carrera	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Gabriela de la C. Rodríguez Rey	

Continúa en la próxima página

Tabla C.3. Continuación de la página anterior

Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión de la información de las carreras.
--

Tabla C.4. Tarea de ingeniería # 7

Tarea	
Número de tarea: 7	Número de Historia de usuario: 7
Nombre de la tarea: Gestionar Grupo	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión de la información de los grupos.	

Tabla C.5. Tarea de ingeniería # 8

Tarea	
Número de tarea: 8	Número de Historia de usuario: 8
Nombre de la tarea: Configurar Distribución	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para la configuración de la distribución.	

Tabla C.6. Tarea de ingeniería # 9

Tarea	
Número de tarea: 9	Número de Historia de usuario: 9
Nombre de la tarea: Distribuir por Facultad	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para realizar la distribución por facultad.	

Tabla C.7. Tarea de ingeniería # 10

Tarea	
Número de tarea: 10	Número de Historia de usuario: 10
Nombre de la tarea: Distribuir por Grupo	

Continúa en la próxima página

Tabla C.7. Continuación de la página anterior

Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para realizar la distribución por grupos.	

Tabla C.8. Tarea de ingeniería # 11

Tarea	
Número de tarea: 11	Número de Historia de usuario: 11
Nombre de la tarea: Gestionar Provincia	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Gabriela de la C. Rodríguez Rey	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión de la información de las provincias.	

Tabla C.9. Tarea de ingeniería # 12

Tarea	
Número de tarea: 12	Número de Historia de usuario: 12
Nombre de la tarea: Gestionar Municipio	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Gabriela de la C. Rodríguez Rey	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión de la información de los municipios.	

Tabla C.10. Tarea de ingeniería # 13

Tarea	
Número de tarea: 13	Número de Historia de usuario: 13
Nombre de la tarea: Reporte vía de ingreso	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestión del reporte de grupos por vía de ingreso.	

Tabla C.11. Tarea de ingeniería # 14

Tarea
Continúa en la próxima página

Tabla C.11. Continuación de la página anterior

Número de tarea: 14	Número de Historia de usuario: 14
Nombre de la tarea: Reporte provincia	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestion del reporte de grupos por provincia.	

Tabla C.12. Tarea de ingeniería # 15

Tarea	
Número de tarea: 15	Número de Historia de usuario: 15
Nombre de la tarea: Reporte general	
Tipo de tarea: Desarrollo	Puntos estimados: 1
Programador responsable: Nelson Castroman Aguiar	
Descripción: Desarrollar los métodos y las interfaces necesarias para la gestion del reporte general.	

Pruebas Unitarias

```
System check identified no issues (0 silenced).
test_crear_curso_invalido (apps.CursoAcademico.tests.CursoAcademicoTests.test_crear_curso_invalido) ... ok
test_crear_curso_valido (apps.CursoAcademico.tests.CursoAcademicoTests.test_crear_curso_valido) ... ok
test_editar_curso_invalido (apps.CursoAcademico.tests.CursoAcademicoTests.test_editar_curso_invalido) ... ok
test_editar_curso_valido (apps.CursoAcademico.tests.CursoAcademicoTests.test_editar_curso_valido) ... ok
test_eliminar_curso_invalido (apps.CursoAcademico.tests.CursoAcademicoTests.test_eliminar_curso_invalido) ... ok
test_eliminar_curso_valido (apps.CursoAcademico.tests.CursoAcademicoTests.test_eliminar_curso_valido) ... ok
test_listar_cursos (apps.CursoAcademico.tests.CursoAcademicoTests.test_listar_cursos) ... ok
test_obtener_curso_invalido (apps.CursoAcademico.tests.CursoAcademicoTests.test_obtener_curso_invalido) ... ok
test_obtener_curso_valido (apps.CursoAcademico.tests.CursoAcademicoTests.test_obtener_curso_valido) ... ok

-----
Ran 9 tests in 0.033s

OK
```

Figura D.1. Pruebas unitarias realizadas a la ruta curso académico

```
System check identified no issues (0 silenced).
test_crear_carrera_invalida (apps.Carrera.tests.CarreraTests.test_crear_carrera_invalida) ... ok
test_crear_carrera_valida (apps.Carrera.tests.CarreraTests.test_crear_carrera_valida) ... ok
test_editar_carrera_invalida (apps.Carrera.tests.CarreraTests.test_editar_carrera_invalida) ... ok
test_editar_carrera_valida (apps.Carrera.tests.CarreraTests.test_editar_carrera_valida) ... ok
test_eliminar_carrera_invalida (apps.Carrera.tests.CarreraTests.test_eliminar_carrera_invalida) ... ok
test_eliminar_carrera_valida (apps.Carrera.tests.CarreraTests.test_eliminar_carrera_valida) ... ok
test_listar_carreras (apps.Carrera.tests.CarreraTests.test_listar_carreras) ... ok
test_obtener_carrera_invalida (apps.Carrera.tests.CarreraTests.test_obtener_carrera_invalida) ... ok
test_obtener_carrera_valida (apps.Carrera.tests.CarreraTests.test_obtener_carrera_valida) ... ok

-----
Ran 9 tests in 0.040s

OK
```

Figura D.2. Pruebas unitarias realizadas a la ruta carrera

```
System check identified no issues (0 silenced).
test_crear_distribucion_activa (apps.Distribucion.tests.DistribucionEstudiantesTests.test_crear_distribucion_activa) ... ok
test_editar_distribucion (apps.Distribucion.tests.DistribucionEstudiantesTests.test_editar_distribucion) ... ok
test_eliminar_distribucion (apps.Distribucion.tests.DistribucionEstudiantesTests.test_eliminar_distribucion) ... ok
test_obtener_distribucion_activa (apps.Distribucion.tests.DistribucionEstudiantesTests.test_obtener_distribucion_activa) ... ok

-----
Ran 4 tests in 0.023s

OK
```

Figura D.3. Pruebas unitarias realizadas a la ruta distribución

```
System check identified no issues (0 silenced).
test_crear_facultad_valida (apps.Facultad.tests.FacultadTests.test_crear_facultad_valida) ... ok
test_editar_facultad_no_existente (apps.Facultad.tests.FacultadTests.test_editar_facultad_no_existente) ... ok
test_editar_facultad_valida (apps.Facultad.tests.FacultadTests.test_editar_facultad_valida) ... ok
test_eliminar_facultad_no_existente (apps.Facultad.tests.FacultadTests.test_eliminar_facultad_no_existente) ... ok
test_eliminar_facultad_valida (apps.Facultad.tests.FacultadTests.test_eliminar_facultad_valida) ... ok
test_facultades_con_carreras (apps.Facultad.tests.FacultadTests.test_facultades_con_carreras) ... ok
test_listar_facultades (apps.Facultad.tests.FacultadTests.test_listar_facultades) ... ok
test_obtener_facultad_no_existente (apps.Facultad.tests.FacultadTests.test_obtener_facultad_no_existente) ... ok
test_obtener_facultad_valida (apps.Facultad.tests.FacultadTests.test_obtener_facultad_valida) ... ok

-----
Ran 9 tests in 0.045s

OK
```

Figura D.4. Pruebas unitarias realizadas a la ruta facultad

```
System check identified no issues (0 silenced).
test_crear_grupo_invalido (apps.Grupo.tests.GrupoTests.test_crear_grupo_invalido) ... ok
test_crear_grupo_valido (apps.Grupo.tests.GrupoTests.test_crear_grupo_valido) ... ok
test_editar_grupo_invalido (apps.Grupo.tests.GrupoTests.test_editar_grupo_invalido) ... ok
test_editar_grupo_valido (apps.Grupo.tests.GrupoTests.test_editar_grupo_valido) ... ok
test_eliminar_grupo_invalido (apps.Grupo.tests.GrupoTests.test_eliminar_grupo_invalido) ... ok
test_eliminar_grupo_valido (apps.Grupo.tests.GrupoTests.test_eliminar_grupo_valido) ... ok
test_listar_grupos (apps.Grupo.tests.GrupoTests.test_listar_grupos) ... ok
test_obtener_grupo_invalido (apps.Grupo.tests.GrupoTests.test_obtener_grupo_invalido) ... ok
test_obtener_grupo_valido (apps.Grupo.tests.GrupoTests.test_obtener_grupo_valido) ... ok

-----
Ran 9 tests in 0.041s

OK
```

Figura D.5. Pruebas unitarias realizadas a la ruta grupo

Pruebas de Aceptación

Tabla E.1. Prueba de aceptación # 5

Caso de prueba de aceptación	
Código: HU2_P1	Historia de usuario: 2
Nombre: Autenticar Usuario	
Descripción: Prueba la funcionalidad de autenticar usuario.	
Condiciones de ejecución: El usuario debe estar registrado en el sistema, el usuario no debe estar autenticado.	
Pasos de ejecución: El usuario ingresa a la página de inicio de sesión, ingresa sus credenciales, el usuario presiona el botón Aceptar.	
Resultados esperados: El usuario accede al sistema.	

Tabla E.2. Prueba de aceptación # 6

Caso de prueba de aceptación	
Código: HU3_P2	Historia de usuario: 3
Nombre: Listar persona	
Descripción: Prueba la funcionalidad de listar persona.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber sido importado el listado de personas.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Configuración, el usuario selecciona la subsección Persona.	
Resultados esperados: El sistema carga una página con el listado de personas	

Tabla E.3. Prueba de aceptación # 7

Caso de prueba de aceptación	
Código: HU3_P3	Historia de usuario: 3
Nombre: Editar persona	
Descripción: Prueba la funcionalidad de editar una persona.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos una persona registrada.	
Pasos de ejecución: El usuario seleccionar el botón Editar, el sistema carga un formulario con los datos de la persona, el usuario modifica los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema edita una persona.	

Tabla E.4. Prueba de aceptación # 8

Caso de prueba de aceptación	
Código: HU3_P4	Historia de usuario: 3
Nombre: Eliminar persona	
Descripción: Prueba la funcionalidad de eliminar una persona.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos una persona registrada.	
Pasos de ejecución: El usuario selecciona el botón de Eliminar, el sistema muestra un cuadro de confirmación, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema elimina una persona.	

Tabla E.5. Prueba de aceptación # 9

Caso de prueba de aceptación	
Código: HU4_P1	Historia de usuario: 4
Nombre: Listar cursos	
Descripción: Prueba la funcionalidad de listar cursos.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos un curso creado.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Configuración, el usuario selecciona la subsección Curso Académico.	
Resultados esperados: El sistema carga una página con el listado de cursos.	

Tabla E.6. Prueba de aceptación # 10

Caso de prueba de aceptación	
Código: HU4_P2	Historia de usuario: 4
Nombre: Registrar curso	
Descripción: Prueba la funcionalidad de registrar un curso.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Añadir, el sistema carga un formulario, el usuario introduce los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema registra un curso.	

Tabla E.7. Prueba de aceptación # 11

Caso de prueba de aceptación	
Código: HU4_P3	Historia de usuario: 4
Nombre: Editar curso	
Descripción: Prueba la funcionalidad de editar un curso.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos un curso creado.	
Pasos de ejecución: El usuario selecciona el botón de Editar, el sistema carga una página con un formulario con los datos cargados, el usuario modifica los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema modifica un curso	

Tabla E.8. Prueba de aceptación # 12

Caso de prueba de aceptación	
Código: HU5_P1	Historia de usuario: 5
Nombre: Listar facultades	
Descripción: Prueba la funcionalidad de listar facultad.	
Condiciones de ejecución: El usuario debe estar autenticado, haber al menos una facultad creada.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Configuración, el usuario selecciona la subsección Facultad.	
Resultados esperados: El sistema carga una página con el listado de facultades.	

Tabla E.9. Prueba de aceptación # 13

Caso de prueba de aceptación	
Código: HU5_P2	Historia de usuario: 5
Nombre: Añadir facultad	
Descripción: Prueba la funcionalidad de registrar facultad.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Añadir, el sistema carga un formulario, el usuario introduce los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema registra una facultad.	

Tabla E.10. Prueba de aceptación # 14

Caso de prueba de aceptación	
Código: HU5_P3	Historia de usuario: 5
Nombre: Editar facultad	
Descripción: Prueba la funcionalidad de editar facultad.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos una facultad creada.	
Pasos de ejecución: El usuario selecciona el botón de Editar, el sistema carga una página con un formulario con los datos cargados, el usuario modifica los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema modifica una facultad	

Tabla E.11. Prueba de aceptación # 15

Caso de prueba de aceptación	
Código: HU6_P1	Historia de usuario: 6
Nombre: listar carrera	
Descripción: Prueba la funcionalidad de listar carrera.	
Condiciones de ejecución: El usuario debe estar autenticado, haber al menos una carrera creada.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Configuración, el usuario selecciona la subsección Carrera.	
Resultados esperados: El sistema carga una página con el listado de carreras.	

Tabla E.12. Prueba de aceptación # 16

Caso de prueba de aceptación	
Código: HU6_P2	Historia de usuario: 6
Nombre: Registrar carrera	
Descripción: Prueba la funcionalidad de registrar carrera.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Añadir, el sistema carga un formulario, el usuario introduce los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema registra una carrera.	

Tabla E.13. Prueba de aceptación # 17

Caso de prueba de aceptación	
Código: HU6_P3	Historia de usuario: 6
Nombre: Editar carrera	
Descripción: Prueba la funcionalidad de editar carrera.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos una carrera creada.	
Pasos de ejecución: El usuario selecciona el botón de Editar, el sistema carga una página con un formulario con los datos cargados, el usuario modifica los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema modifica la carrera.	

Tabla E.14. Prueba de aceptación # 18

Caso de prueba de aceptación	
Código: HU7_P1	Historia de usuario: 7
Nombre: Listar grupo	
Descripción: Prueba la funcionalidad de listar grupo.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos un grupo creado.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Configuración, el usuario selecciona la subsección Grupo.	
Resultados esperados: carga una página con el listado de grupos.	

Tabla E.15. Prueba de aceptación # 19

Caso de prueba de aceptación	
Código: HU7_P2	Historia de usuario: 7
Nombre: Crear grupo	
Descripción: Prueba la funcionalidad crear grupo.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haberse configurado una distribución.	
Pasos de ejecución: El usuario selecciona el botón de Crear Grupo, el sistema carga un formulario, el usuario introduce los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema crea los grupos.	

Tabla E.16. Prueba de aceptación # 20

Caso de prueba de aceptación	
Código: HU7_P3	Historia de usuario: 7
Nombre: Editar grupo	
Descripción: Prueba la funcionalidad de editar grupo.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos un grupo creado.	
Pasos de ejecución: El usuario selecciona el botón de Editar, el sistema carga una página con un formulario con los datos cargados, el usuario modifica los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema modifica un grupo.	

Tabla E.17. Prueba de aceptación # 21

Caso de prueba de aceptación	
Código: HU8_P1	Historia de usuario: 8
Nombre: Configurar distribución	
Descripción: Prueba la funcionalidad de configurar distribución.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Configuración, el usuario selecciona la subsección Distribución, el usuario rellena los datos de la tabla.	

Continúa en la próxima página

Tabla E.17. Continuación de la página anterior

Resultados esperados: El sistema guarda la distribución configurada.

Tabla E.18. Prueba de aceptación # 22

Caso de prueba de aceptación	
Código: HU9_P1	Historia de usuario: 9
Nombre: Distribución por facultad	
Descripción: Prueba la funcionalidad de distribuir por facultad.	
Condiciones de ejecución: El usuario debe estar autenticado, debe tener creado facultades, carreras, debe haber importado personas, debe haber configurado la distribución.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Ingreso, el usuario selecciona la subsección Distribución por Facultad, el usuario selecciona la carrera que va a distribuir, el usuario presiona el botón Distribuir.	
Resultados esperados: El sistema distribuye las personas por facultad	

Tabla E.19. Prueba de aceptación # 23

Caso de prueba de aceptación	
Código: HU10_P1	Historia de usuario: 10
Nombre: Distribución por grupo	
Descripción: Prueba la funcionalidad de distribuir por grupo.	
Condiciones de ejecución: El usuario debe estar autenticado, debe tener creado facultades, carreras, debe haber importado personas, debe haber configurado la distribución, debe haberse realizado la distribución por facultad.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Ingreso, el usuario selecciona la subsección Distribución por Grupo, el usuario selecciona la facultad que va a distribuir, el usuario presiona el botón Distribuir.	
Resultados esperados: El sistema distribuye las personas por grupo	

Tabla E.20. Prueba de aceptación # 24

Caso de prueba de aceptación	
Código: HU11_P1	Historia de usuario: 11
Nombre: Listar provincia	

Continúa en la próxima página

Tabla E.20. Continuación de la página anterior

Descripción: Prueba la funcionalidad de listar las provincias.
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos una provincia creada.
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Configuración, el usuario selecciona la subsección Nomenclador, el usuario selecciona la subsubsección Provincia.
Resultados esperados: El sistema lista las provincias.

Tabla E.21. Prueba de aceptación # 25

Caso de prueba de aceptación	
Código: HU11_P2	Historia de usuario: 11
Nombre: Añadir provincia	
Descripción: Prueba la funcionalidad de añadir provincia.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Añadir, el sistema carga una página un formulario, el usuario introduce los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema registra una provincia.	

Tabla E.22. Prueba de aceptación # 26

Caso de prueba de aceptación	
Código: HU11_P3	Historia de usuario: 11
Nombre: Editar provincia	
Descripción: Prueba la funcionalidad de editar una provincia.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos una provincia creada.	
Pasos de ejecución: El usuario selecciona el botón Modificar, el sistema carga una página con un formulario con los datos, el usuario modifica los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema modifica una provincia.	

Tabla E.23. Prueba de aceptación # 27

Caso de prueba de aceptación	
Código: HU11_P4	Historia de usuario: 11
Nombre: Eliminar provincia	
Descripción: Prueba la funcionalidad de eliminar provincia.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos una provincia creada.	
Pasos de ejecución: El usuario selecciona el botón de Eliminar, el sistema muestra un cuadro de confirmación, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema elimina una provincia	

Tabla E.24. Prueba de aceptación # 28

Caso de prueba de aceptación	
Código: HU12_P1	Historia de usuario: 12
Nombre: Listar municipio	
Descripción: Prueba la funcionalidad de listar los municipios.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos un municipio creado.	
Pasos de ejecución: El usuario selecciona en el menú lateral la sección Configuración, el usuario selecciona la subsección Nomenclador, el usuario selecciona la subsubsección Municipio.	
Resultados esperados: El sistema lista los municipios.	

Tabla E.25. Prueba de aceptación # 29

Caso de prueba de aceptación	
Código: HU12_P2	Historia de usuario: 12
Nombre: Añadir municipio	
Descripción: Prueba la funcionalidad de añadir municipio.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Añadir, el sistema carga una página un formulario, el usuario introduce los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema registra un municipio.	

Tabla E.26. Prueba de aceptación # 30

Caso de prueba de aceptación	
Código: HU12_P3	Historia de usuario: 12
Nombre: Editar municipio	
Descripción: Prueba la funcionalidad de editar un municipio.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos un municipio creado.	
Pasos de ejecución: El usuario selecciona el botón Modificar, el sistema carga una página con un formulario con los datos, el usuario modifica los datos necesarios y verifica que sean correctos, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema modifica un municipio.	

Tabla E.27. Prueba de aceptación # 31

Caso de prueba de aceptación	
Código: HU12_P4	Historia de usuario: 12
Nombre: Eliminar municipio	
Descripción: Prueba la funcionalidad de eliminar municipio.	
Condiciones de ejecución: El usuario debe estar autenticado, debe haber al menos un municipio creado.	
Pasos de ejecución: El usuario selecciona el botón de Eliminar, el sistema muestra un cuadro de confirmación, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema elimina un municipio	

Tabla E.28. Prueba de aceptación # 32

Caso de prueba de aceptación	
Código: HU13_P1	Historia de usuario: 13
Nombre: Lista reporte vía de ingreso	
Descripción: Prueba la funcionalidad de listar reporte vía de ingreso.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona la sección Reporte, el usuario selecciona la subsección Vía de ingreso.	
Resultados esperados: El sistema carga la lista del reporte.	

Tabla E.29. Prueba de aceptación # 33

Caso de prueba de aceptación	
Código: HU13_P2	Historia de usuario: 13
Nombre: Exportar reporte vía de ingreso	
Descripción: Prueba la funcionalidad de exportar reporte vía de ingreso.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Exportar, el sistema muestra un cuadro de confirmación, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema exporta el reporte vía de ingreso en formato excel.	

Tabla E.30. Prueba de aceptación # 34

Caso de prueba de aceptación	
Código: HU14_P1	Historia de usuario: 14
Nombre: Lista reporte provincia	
Descripción: Prueba la funcionalidad de listar reporte provincia.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona la sección Reporte, el usuario selecciona la subsección Provincia.	
Resultados esperados: El sistema carga la lista del reporte.	

Tabla E.31. Prueba de aceptación # 35

Caso de prueba de aceptación	
Código: HU14_P2	Historia de usuario: 14
Nombre: Exportar reporte provincia	
Descripción: Prueba la funcionalidad de exportar reporte provincia.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Exportar, el sistema muestra un cuadro de confirmación, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema exporta el reporte provincia en formato excel.	

Tabla E.32. Prueba de aceptación # 36

Caso de prueba de aceptación	
Código: HU15_P1	Historia de usuario: 15
Nombre: Lista reporte general	
Descripción: Prueba la funcionalidad de listar reporte general.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona la sección Reporte, el usuario selecciona la subsección General.	
Resultados esperados: El sistema carga la lista del reporte.	

Tabla E.33. Prueba de aceptación # 37

Caso de prueba de aceptación	
Código: HU15_P2	Historia de usuario: 15
Nombre: Exportar reporte general	
Descripción: Prueba la funcionalidad de exportar reporte general.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Exportar, el sistema muestra un cuadro de confirmación, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema exporta el reporte general en formato excel.	

Tabla E.34. Prueba de aceptación # 38

Caso de prueba de aceptación	
Código: HU3_P1	Historia de usuario: 3
Nombre: Importar listado de personas	
Descripción: Prueba la funcionalidad de importar listado de personas.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Importar personas, el sistema muestra una ventana para buscar la ubicación del archivo, el usuario selecciona el archivo, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema importa el listado de personas.	

Tabla E.35. Prueba de aceptación # 39

Caso de prueba de aceptación	
Código: HU3_P5	Historia de usuario: 3
Nombre: Exportar plantilla vacía	
Descripción: Prueba la funcionalidad de exportar plantilla vacía.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Exportar plantilla, el sistema muestra un cuadro de confirmación, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema exporta la plantilla vacía en formato excel.	

Tabla E.36. Prueba de aceptación # 40

Caso de prueba de aceptación	
Código: HU3_P6	Historia de usuario: 3
Nombre: Exportar distribución	
Descripción: Prueba la funcionalidad de exportar distribución.	
Condiciones de ejecución: El usuario debe estar autenticado.	
Pasos de ejecución: El usuario selecciona el botón de Exportar distribución, el sistema muestra un cuadro de confirmación, el usuario presiona el botón Aceptar.	
Resultados esperados: El sistema exporta la distribución en formato excel.	

APÉNDICE F

Indicadores

**Entrevista para el desarrollo del Sistema de Gestión para el ingreso y distribución de los nuevos estudiantes de la Universidad de las Ciencias Informáticas (UCI)****Fecha: 23 de junio de 2024****Nombre de los entrevistadores:** Est. Gabriela de la C. Rodríguez Rey, Est. Nelson Castroman Aguiar**Centro de Estudio:** Universidad de las Ciencias Informáticas (UCI)**Nombre del entrevistado:** Ing. Jorge Arias Sojo**Centro de Trabajo:** Dirección de Informatización de la Universidad de las Ciencias Informáticas (UCI)**Objetivo:** Obtener información sobre el proceso de ingreso de los estudiantes en la universidad para que el sistema de gestión para el ingreso y distribución de los nuevos estudiantes cumpla con el protocolo requerido para su puesta en funcionamiento.**Preguntas y Respuestas**

1. ¿Cuáles son los principales procesos o tareas que este sistema debe gestionar?

El sistema se centra principalmente en la distribución de estudiantes a facultades y grupos, y la generación de reportes sobre las distribuciones realizadas. También incluye tareas administrativas relacionadas con la gestión de usuarios. Y se gestionan personas (estudiantes), cursos académicos, facultades, carreras y grupos.

2. ¿Qué problemas específicos enfrenta actualmente y cómo espera que el desarrollo de este nuevo sistema los resuelva?

El sistema viene a resolver varios problemas clave. En primer lugar, la distribución de estudiantes a facultades y grupos puede ser un proceso complicado, especialmente cuando hay muchas variables a considerar. El algoritmo genético ayudaría a hacer este proceso más rápido y preciso, reduciendo la carga manual. Además, el sistema busca centralizar toda la información en un solo lugar, lo que mejoraría la accesibilidad y el control. Esto también facilita la automatización de tareas que actualmente se hacen de manera manual lo que nos permite ahorrar tiempo y minimizar errores.

3. ¿Qué factores se deben tener en cuenta a la hora de hacer la distribución?

Al hacer la distribución de estudiantes, es importante tener en cuenta el género, para asegurar un balance adecuado en los grupos, el índice académico, para asignar a los estudiantes a grupos adecuados a su rendimiento, y la provincia de procedencia, para considerar la proximidad y evitar concentraciones desproporcionadas de estudiantes de la misma región en una misma facultad y/o grupo.