Nota de Aceptación

Aprobado por el Comité de Trabajo de Grado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana para optar el título de Ingeniero de Sistemas y Computación.

**Hernán Camilo Rocha Niño**
Decano de la Facultad de Ingeniería y Ciencias

**Gerardo M. Sarria M.**
Director Carrera Ingeniería Sistemas y Computación.

**Juan Carlos Martínez Arias**
Director Trabajo de Grado

**Gloria Inés Álvarez V.**
Jurado 1

**Andrés Adolfo Navarro Newball**
Jurado 2

# Acta de Correcciones al Proyecto de Grado
# Ingeniería de Sistemas y Computación

**Fecha:** 02/07/2021

**Autores:** Laura Arango Mayor y Verónica Tofiño Rodríguez

**Nombre del Proyecto de Grado:** Prototipo funcional de un Simulador de Historias Clínicas Nutricionales enfocado hacia los estudiantes

**Director:** Mgtr. Juan Carlos Martínez Arias

Como indica el artículo 2.27 de las Directrices de Trabajo de Grado, he verificado que las estudiantes indicadas arriba, han implementado todas las correcciones que los Jurados del Proyecto de Grado definieron que se efectuaran, como consta en el Acta de Calificación correspondiente.

Juan Carlos Martínez Arias
Director del Proyecto de Grado

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias.
Ingeniería de Sistemas y Computación.

# Prototipo funcional de un Simulador de Historias Clínicas Nutricionales enfocado hacia los estudiantes

Laura Arango Mayor
Verónica Tofiño Rodríguez

Director: Mgtr. Juan Carlos Martínez Arias
Co-Director: Dr. Gerardo M. Sarria M.

11 de Junio de 2021

# Resumen

La simulación es una técnica de aprendizaje cada vez más utilizada en diferentes campos. Su uso en la educación ha ayudado a que los estudiantes desarrollen mejores habilidades prácticas y comunicativas. Particularmente en el área de la salud, la simulación tiene gran acogida puesto que ayuda a que los estudiantes practiquen los conceptos que van aprendiendo en los semestres sin poner en riesgo la vida de los pacientes, lo que a su vez hace que los estudiantes se sientan más confiados de lo que van a realizar cuando salgan al mundo laboral.

En el área de la nutrición sin embargo, casi no existen herramientas que ayuden a los estudiantes a poner en práctica lo aprendido, la mayoría de sistemas existentes se enfocan en el área de la medicina, por lo que las personas de nutrición deben utilizar estas herramientas que no se acogen completamente en sus necesidades. Es por esto que este trabajo tiene como finalidad el desarrollo e implementación de un prototipo funcional que simule la creación y gestión de una historia clínica nutricional (HCN, de ahora en adelante) para los estudiantes de la carrera de Nutrición y Dietética de la Pontificia Universidad Javeriana Cali.

La implementación de este prototipo se realizó siguiendo los pasos del proceso de desarrollo de software (abreviado como SW) y de diferentes tecnologías y enfoques actuales para el desarrollo de SW. Como el sistema se centra en el aprendizaje y evaluación hacía los estudiantes, cuenta con diferentes elementos que, se consideran, ayudarán al estudiante de nutrición a gestionar el llenado de una historia clínica nutricional de forma acertada.

**Palabras Clave**: Nutrición y Dietética, Historias Clínicas Nutricionales, Simulación, Simulación educativa.

# Abstract

Simulation is a learning technique increasingly used in different fields. It use in education has helped students to develop better practical and communicative skills. Particularly in the health area, simulation is very popular because it helps students practice the concepts they are learning in the semesters without putting patients' lives at risk, which in turn makes students feel more confident about what they are going to do when they go out into the professional world.

In the area of nutrition however, there are almost no tools that help students to put into practice what they have learned, most existing systems are focused on the area of medicine, so people of nutrition must use these tools that are not fully embraced in their needs.That is why this work aims to develop and implement a functional prototype that simulates the creation and management of a nutritional health record (HCN, from now on) for students of the career of Nutrition and Dietetics of the Pontificia Universidad Javeriana Cali.

The implementation of this prototype was done following the steps of the software development process (abbreviated as SW) and different current technologies and approaches for SW development. As the system is focused on learning and evaluation for the students, it has different elements that, it is considered, will help the nutrition student to manage the filling of a nutritional clinical history in an accurate way.

**Key Words**: Nutrition and Dietetics, Nutritional Assessment, Simulation, Educational Simulation.

# Índice general

# Introducción

El costo que tenía introducir la simulación como método de aprendizaje, inicialmente, era alto por las limitaciones tecnológicas de la época [Lat10]. Sin embargo, en la actualidad este costo se ha reducido considerablemente llegando a ser un método de aprendizaje viable para la mayoría de las instituciones educativas. El área de la salud es uno de los campos que más ha investigado y explotado a la simulación como método de enseñanza y aprendizaje por la cantidad de habilidades activas y pasivas que le puede proveer a los estudiantes. El uso de la simulación le permite a los estudiantes poder practicar lo que van aprendiendo de forma teórica a lo largo de los semestres de forma segura puesto que admite que los estudiantes se puedan equivocar y aprender de sus errores sin poner en riesgo la vida de los pacientes. Además de esto, les ayuda a mejorar sus habilidades comunicativas y a sentirse más seguros al momento de tener que interactuar con pacientes reales.

Es por esto que, en pro de preparar profesionales cada vez mejor capacitados para el mundo laboral, las instituciones educativas han ido adicionando a su plan de estudios herramientas de simulación para la capacitación profesional de los estudiantes del área de la salud.

En este documento se presentan los pasos definidos para el ciclo de desarrollo de un software (abreviado como SDLC en sus siglas inglesas) enfocados al desarrollo e implementación de un prototipo funcional, que simule la creación y gestión de una historia clínica nutricional (HCN, de ahora en adelante) para los estudiantes de la carrera de Nutrición y Dietética de la Pontificia Universidad Javeriana Cali. La implementación de este prototipo se realizó utilizando tecnologías, herramientas y enfoques actuales del desarrollo de software. Además, al ser un sistema con enfoque hacía el aprendizaje de los estudiantes, el sistema cuenta con elementos que, se consideran, ayudarán al estudiante de nutrición a gestionar el llenado de una historia clínica nutricional de forma acertada.

Capítulo 1

# Descripción del Problema

## 1.1. Planteamiento del Problema

En la actualidad se cuentan con diversidad de métodos para el aprendizaje y uno de los más útiles es la simulación, siendo empleada tanto con fines educacionales como evaluativos. La Educación Clínica Alternativa (ACE, por sus siglas en inglés) define la simulación como la representación artificial de un proceso del mundo real para lograr metas educativas a través del aprendizaje experimental. Es por tanto que se infiere que *"la simulación aplicada en el área de la salud es un buen complemento del proceso del docente que facilita, pero no sustituye, la interacción del estudiante con la realidad de los servicios de salud"* [SA95]. Algunos ejemplos de sistemas desarrollados con el fin de replicar en un entorno virtual tareas de la vida real son aquellos que pretenden gestionar y analizar historias clínicas nutricionales, apoyando así, estudiantes universitarios en cuyas carreras sea relevante el tema. Programas como NutPlan, desarrollado por la Universidad de Belgrado [Gur+10], Nutrium, programa aliado de más de veinticinco universidades a nivel mundial [Nut19] y el SFC de la Universidad Católica de la Santísima Concepción [Tro+18], son ejemplos de sistemas desarrollados para que las universidades puedan brindar un mejor servicio de enseñanza a sus estudiantes. Los anteriores sistemas reconocen la utilidad de la simulación y los beneficios que acarrea para estudiantes y profesores.

La carrera de Nutrición y Dietética de la Pontificia Universidad Javeriana, creada en el 2016, tiene como objetivo *desarrollar competencias en el estudiante para establecer interacciones, planear, desarrollar y evaluar las acciones en materia de la nutrición y la dietética en los ámbitos hospitalarios, de la industria de alimentos, la nutrición pública y de la gastronomía* [Pon16]. Durante los semestres se ven diferentes materias para que los estudiantes vayan conociendo cuál de los diferentes campos de acción, mencionados con anterioridad, quieren tomar una vez se gradúen. Uno de los principales enfoques de la carrera es el ambiente hospitalario, es decir, que los profesionales se ocupen de la prevención, diagnóstico y tratamiento de los cambios nutricionales y metabólicos relacionados con enfermedades agudas o crónicas y con condiciones causadas por un exceso o déficit de nutrientes. Para esto se utilizan las historias clínicas nutricionales.

Durante años, la carrera de Nutrición y Dietética ha impartido sobre los estudiantes todo el ámbito teórico que gira en torno a la elaboración y desarrollo de las historias clínicas nutricionales, pero con el fin de formar mejores profesionales, la carrera constantemente busca nuevas formas de enseñanza sobre sus estudiantes mediante las mejores metodologías disponibles, con las que puedan impartir

conocimientos íntegros mediante una educación que se beneficia y hace uso responsable de las herramientas tecnológicas al alcance de la institución. Por tanto, la universidad está constantemente considerando nuevos tipos de herramientas tecnológicas que aporten al proceso de aprendizaje de los estudiantes. Dentro de dichas herramientas se encuentra un sistema que simule un entorno didáctico en el cual los estudiantes se familiaricen con la elaboración y manejo de las historias clínicas nutricionales.

Desde la pedagogía, contar con un software de simulación óptimo aporta al proceso de aprendizaje de los estudiantes. Diversos estudios han mostrado que, en comparación con la clase tradicional, los programas multimediales pueden ayudar al estudiante a aprender los conceptos de forma más rápida. Algunos estiman que se puede ahorrar hasta un 80 por ciento de tiempo en el aprendizaje en entornos simulados [Min04]. Este tipo de sistemas les ofrecen a los estudiantes la oportunidad de no solo quedarse con la teoría aprendida en clase, sino también prepararse mejor para cuando deban poner en práctica sus conocimientos en un ambiente laboral, permitiéndoles cometer errores y ser guiados hacia la corrección de estos mientras estudian. Con eso se logra que cuando estén laborando y se encuentren en situaciones como las simuladas, sepan cómo proceder de la mejor manera posible, puesto que tendrán los conocimientos y la experiencia para lidiar con esa clase de problemas.

### 1.1.1. Formulación

¿Cómo diseñar e implementar un software de manejo de las historias clínicas nutricionales para apoyar el proceso de aprendizaje académico-práctico de los estudiantes de la carrera de Nutrición y Dietética de la Pontificia Universidad Javeriana Cali?

### 1.1.2. Sistematización

- ¿Cuáles son los aspectos más importantes de las historias clínicas nutricionales y cómo la simulación se puede implementar para ayudar al aprendizaje en el área de la salud?
- ¿Cómo se establecerán los requisistos del sistema a desarrollar?
- ¿Cómo será el diseño y la construcción del software?
- ¿Cómo se implementará el prototipo funcional del sistema?
- ¿Cómo se evaluará el prototipo de software desarrollado?

## 1.2. Objetivos

### 1.2.1. Objetivo General

Diseñar e implementar un prototipo funcional de manejo de las historías clínicas nutricionales para apoyar el proceso de aprendizaje académico-práctico de los estudiantes de la carrera de Nutrición y Dietética de la Pontificia Universidad Javeriana Cali.

### 1.2.2. Objetivos Específicos

- Identificar y analizar los aspectos más importantes de las historias clínicas nutricionales y el uso de la simulación para el aprendizaje de los estudiantes en el área de la salud.
- Establecer y especificar los requisitos del sistema a desarrollar, enfocándose en la construcción de un módulo para el aprendizaje de los estudiantes con diferentes elementos que aporten a esto.
- Diseñar la arquitectura y los componentes del software a desarrollar, así como los aspectos de simulación que se integrarán a la solución.
- Desarrollar un prototipo funcional que cumpla con los requisitos establecidos para el sistema.
- Evaluar el sistema mediante pruebas funcionales (de unidad y de componentes) y de usabilidad.

Capítulo 2

# Desarrollo del Proyecto

## 2.1. Marco de Referencia

### 2.1.1. Áreas Temáticas

Las áreas temáticas que se trabajaron fueron [ACM12]:

- Applied computing → Life and medical sciences → Health care information systems
- Applied computing → Education → Interactive learning environments
- Applied computing → Education → Computer-assisted instruction
- Software and its engineering → Software creation and management →Designing software
- Software and its engineering → Software organization and properties → Software system structures → Software architectures
- Software and its engineering → Software creation and management → Software verification and validation → Software prototyping

### 2.1.2. Marco Teórico

Este proyecto de grado se enfocó en la construcción de un prototipo funcional de un software que simule historias clínicas nutricionales como apoyo para el aprendizaje de los estudiantes de la carrera de Nutrición y Dietética de la Pontificia Universidad Javeriana Cali. Es por eso que a continuación se presentan los conceptos de los temas más relevantes que se tuvieron en cuenta en el desarrollo del prototipo funcional del sistema.

#### 2.1.2.1. Proceso de cuidado nutricional

En el 2003 la Asociación Americana de Dietética (ADA, por sus siglas en inglés) estableció los lineamientos del Proceso de Cuidado Nutricional (PCN) el cual describe como un profesional de la nutrición debe proveer cuidado a los pacientes. *El PCN es un enfoque sistemático para proporcionar una atención nutricional de alta calidad. Este consta de cuatro pasos distintos e interrelacionados*[Aca]:

1. La evaluación del estado nutricional: En este paso se hace toda la recolección de datos y documentos para poder hacer un correcto diagnóstico de nutricional del paciente. En este paso es que el profesional utiliza un formato de historia clínica nutricional para recolectar dicha información.

2. El diagnóstico nutricional: En este paso es que se listan las enfermedades nutricionales que presenta el paciente acorde al llenado realizado en el primer paso, para así definir el diagnóstico del paciente.

3. Intervención nutricional: En este paso el profesional le informa el tratamiento, junto con el plan nutricional, que el paciente debe seguir para mejorar su estado nutricional.

4. Monitoreo nutricional y evaluación: Es el paso final del PCN. En este el profesional evalúa el seguimiento del paciente con el plan recitado en el paso anterior.

El seguimiento de estos pasos ayuda al profesional a realizar un diagnóstico más acertado de la persona y poder así ayudarlo de forma correcta.

#### 2.1.2.2. Historias clínicas nutricionales

Como se explicó anteriormente, la historia clínica nutricional (HCN) es el primer paso para que el profesional pueda diagnosticar y tratar el estado nutricional de una persona. *La historia clínico-nutriológica es un conjunto de documentos y herramientas que permiten reunir información mediante una entrevista con el paciente y, en caso necesario, con sus familiares* [SH10]. Mediante la recolección de los datos suministrados se pueden identificar los problemas relacionados con la nutrición del paciente al igual que sus causas y poder plantear así una correcta solución.

La historia clínica nutricional se encarga de recolectar datos del paciente desde diferentes enfoques para así poder dar un diagnóstico más preciso. Por esto se emplean diferentes indicadores y exámenes que ayudan al profesional a recolectar la mayor cantidad de información pertinente para el diagnóstico. Existen diferentes formas de dividir una HCN, sin embargo, se identificaron seis módulos o partes generales de esta. Es importante recalcar que, todos los indicadores y exámenes realizados al paciente se encuentran dentro de cuatro módulos generales llamados el ABCD por sus siglas capitales; estos son los indicadores Antropométricos, los indicadores Bioquímicos, los indicadores Clínicos y los indicadores Dietéticos.

A continuación, se presenta en la figura 2.1 un diagrama de bloques que conforma de manera macro como se compone una historia clínica nutricional.

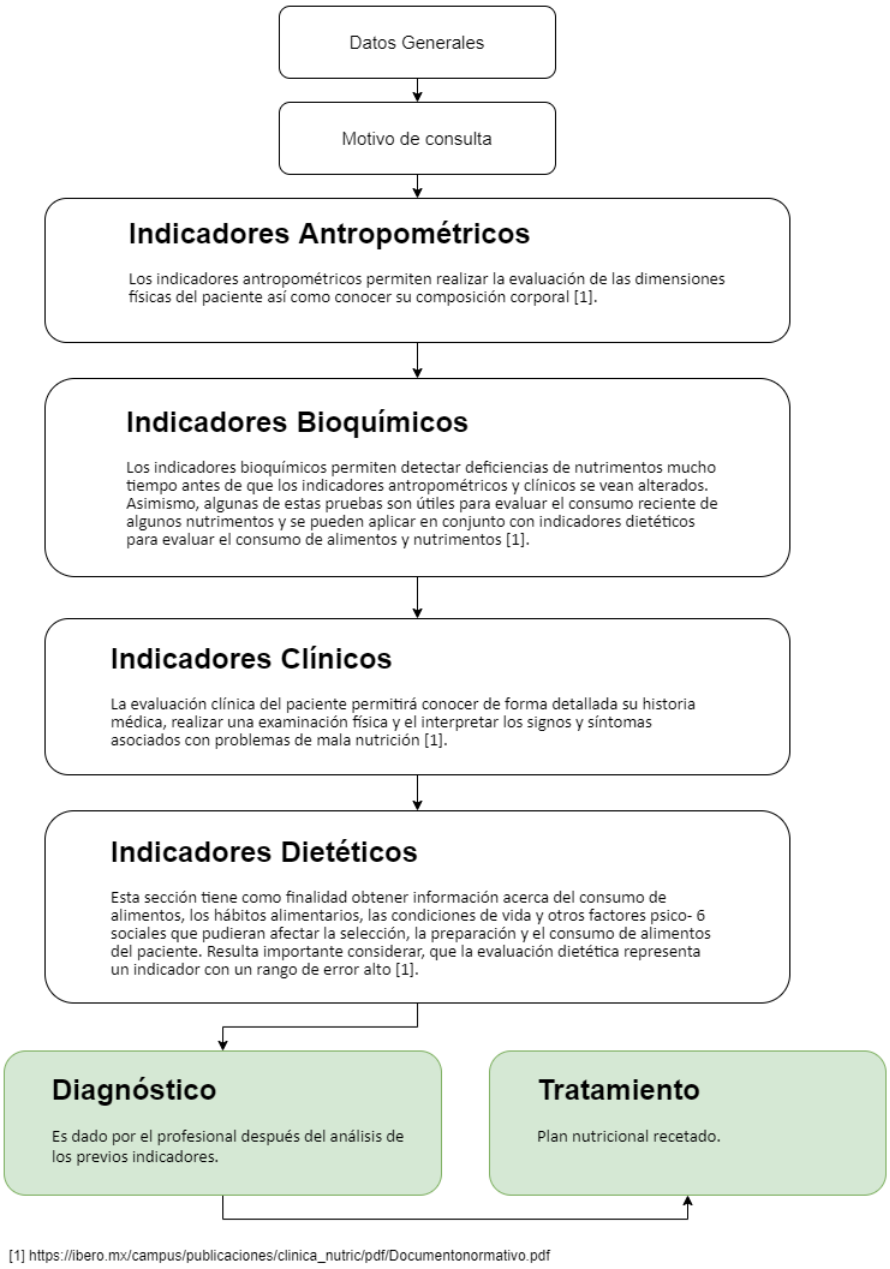



Figura 2.1: Diagrama de bloques de la composición de un formato de HCN

En la figura 2.1 se muestran seis módulos identificados para un formato de la HCN y se agregan dos campos (resaltados en verde) que serán explicados más adelante. Una explicación más detallada de estos módulos es:

- Datos generales: En esta sección se realiza la toma de datos personales del paciente como

nombre y apellidos, fecha de nacimiento, género con el que se identifica, sexo, edad, EPS, teléfono, ocupación y estado civil. También se identifica la HCN con un numero único y se escribe la fecha de la consulta.

- Motivo de la consulta: En este módulo el paciente debe explicarle, de la forma más detalladamente posible al profesional, cual es la razón que lo llevó a generar la consulta con el nutricionista. En este paso el profesional se limita a escuchar de forma atenta al paciente y recolecta la mayor cantidad de información posible para evaluarla después, con la toma de los demás datos.

- Datos antropométricos: Son procedimientos no invasivos que se realizan en el paciente para conocer los datos físicos. Mediante la recolección de estos datos se dan a conocer el peso, la talla, la estructura y el índice de masa corporal (IMC) entre otras medidas físicas del paciente. La recolección de estos datos ayuda al profesional a conocer el estado de salud relacionándolo con su peso, edad y talla. *Es un aspecto muy importante a la hora de elaborar una dieta o plan nutricional, pues permite cuantificar las reservas corporales del organismo y, por lo tanto, detectar problemas nutricionales como situaciones de sobrepeso y obesidad, en las que existe un exceso de masa grasa o, por el contrario, desnutrición, situación en la que tanto la masa grasa como la masa muscular podrían verse disminuidas* [Jor16].

  Algunas de las medidas que se toman y calculan son:

  - Peso: Es un indicador global de la masa corporal del cuerpo de una persona. Es uno de los mejores indicadores para conocer el estado nutricional del paciente.
  - Peso de referencia: Se establece mediante una tabla de referencia en función del peso, talla y complexión del individuo.
  - Peso usual: Si el paciente se encuentra enfermo es una variable más confiable que el peso de referencia.
  - % cambio de peso: Sirve como alternativa para evaluar los cambios de peso en el paciente. Es importante resaltar que la pérdida de peso y su importancia recae en el tiempo que se realice. No es igual de significativo perder peso en un lapso de seis meses a perder peso en un lapso de una semana.
  - Pliegue tricipal: El pliegue tricipal mide el espesor del tejido adiposo del tríceps. Junto con la toma de datos de otros pliegues ayuda a estimar la grasa corporal en el cuerpo.
  - Pliegue subescapular: El pliegue subescapular mide el espesor del tejido adiposo debajo del omoplato. Su importancia es la misma explicada en el pliegue tricipal.
  - Perímetro braquial: Medida de la circunferencia de la parte superior del brazo. Es un indicador de la pérdida de masa muscular en el brazo.
  - Perímetro abdominal: Medida de la circunferencia de la zona abdominal en el punto medio del abdomen. Ayuda a detectar factores de riesgo cardiovasculares acorde a la grasa acumulada en la zona.

- Talla: Es el parámetro para determinar la longitud de una persona. Su importancia recae en que, en niños es utilizada para evaluar el crecimiento y desarrollo de estos, mientras que en los adultos ayuda a calcular otros índices importantes y confiables como el índice de masa corporal.
- Estructura: La estructura o complexión se refiere a la amplitud o peso del esqueleto humano. Esta medida se encuentra por medio de una ecuación matemática que utiliza la talla y la circunferencia del carpo (hueso en la muñeca).
- Índice de Masa Corporal (IMC): Es la razón matemática que asocia la masa y la talla del individuo. Es un indicador confiable para calcular posibles problemas de salud. Acorde a los valores que muestre puede asociarse enfermedades mortales como cáncer, diabetes e infecciones respiratorias y gastrointestinales.

- Datos bioquímicos: La toma de estos datos se realiza mediante exámenes médicos. Los indicadores bioquímicos ayudan a detectar deficiencias en los nutrientes de forma temprana, es decir, antes de que se presenten signos y síntomas clínicos. Algunas veces ayuda a comprobar teorías de los nutricionistas sobre causas de la malnutrición de un paciente. *Asimismo, algunas de estas pruebas son útiles para evaluar el consumo reciente de algunos nutrimentos y se pueden aplicar en conjunto con indicadores dietéticos para evaluar el consumo de alimentos y nutrimentos* [SSP04].

  Algunos exámenes que se pueden realizar para comprobar o verificar cierta sospecha de un nutricionista son:

  - Toma de suero/plasma: La concentración muestra la ingestión dietética reciente.
  - Toma de un nutrimento específico: Sirve para detectar el numero de eritrocitos en la sangre y así saber si la persona sufre de anemia, deshidratación, desnutrición o leucemia.
  - Orina: Ayuda a saber la reciente ingestión de líquidos y alimentos.

  Sin embargo, la toma de estos exámenes debe ser especifica y justificada dado que son muy costosos de realizar, en la mayoría de los casos son métodos invasivos y se debe ser muy preciso en el análisis de la muestra dado que se puede contaminar fácilmente y dañar los resultados [CC04]

- Datos clínicos: Estos datos ayudan a construir la historia médica de los pacientes. Con la recolección de estos datos se pueden conocer enfermedades (actuales y pasadas) que sufra la persona, al igual que medicamentos que pueden estar causando un desbalance en el cuerpo. Es importante recolectar esta información porque el padecimiento de enfermedades y la toma de medicamentos para estos puede ser lo que está afectando los nutrimentos de la persona y causando un des balance en la persona. Por otro lado, *determinar la presencia de síntomas o problemas gastrointestinales y bucales que puedan afectar el consumo, digestión o absorción de nutrimentos como por ejemplo si es edéntulo, si presenta vómito, diarrea o estreñimiento*

[SSP04]. También se debe realizar un examen físico cualitativo donde se evalúa el aspecto de cabello, piel, ojos, entre otros al igual que la actitud del paciente que ayudan a detectar si el paciente está mostrando síntomas de alguna deficiencia o exceso de nutrientes.

Es importante a su vez, en el caso de las mujeres, preguntarles por antecedentes ginecológicos dado que los ciclos menstruales, el consumo de píldoras hormonales o los embarazos puede afectar los nutrimentos de una mujer, afectar su peso corporal y causar desbalance general en el cuerpo.

- Datos dietéticos: La toma de estos datos se realiza para saber el estilo de vida actual del paciente. Conocer, por ejemplo, la frecuencia con la que consume ciertos tipos de alimentos que pueden influenciar en su nutrición, las actividades que realiza y sus hábitos alimentarios generales. Como este indicador se basa en el diario vivir de la persona puede resultar con dificultad el realizar conclusiones precisas dado que los pacientes pueden olvidar lo que han ingerido o han realizado con anterioridad. Es por esto que, para términos de mayor exactitud, existe una encuesta llamada "Recordatorio de 24 horas". Aquí se escribe el registro de comidas de las últimas 24 horas (desayuno, refrigerio, almuerzo, cena, y refrigerio nocturno) previas a la consulta, si este examen se toma en reiteradas ocasionadas puede ayudar a establecer el consumo habitual de las personas. En general, en esta sección se intenta que los exámenes y preguntas que se realizan sean lo más específicas y cercanas a la fecha de la consulta posible para disminuir el rango de error.

En este sistema en particular, como se puede evidenciar en la figura 2.1, se adicionan dos campos o secciones más dentro del formato de la historia clínica nutricional por términos de aprendizaje para los estudiantes. Estos campos adicionados son:

- Diagnóstico: Pertenece al segundo paso del proceso de cuidado nutricional. En este campo se explica de forma global la interpretación de los datos y exámenes recolectados en cada uno de los indicadores para poder así conocer el estado nutricional del paciente de forma objetiva. Una vez se tenga esto se pasa a utilizar la Clasificación Estadística Internacional de Enfermedades y Problemas Relacionados con la Salud (CIE), versión en español de la versión en inglés ICD, para clasificar con códigos las enfermedades, signos, síntomas, circunstancias sociales, entre otros, del paciente. Utilizar esta estandarización ayuda a que pueda ser interpretada por cualquier nutricionista al ser aceptado a nivel mundial. Actualmente se utiliza la CIE-10 (decima versión del CIE) sin embargo a partir del 2022 se utilizará la CIE-11.

- Tratamiento: Pertenece al tercer paso del PCN. En este campo el profesional receta el plan nutricional que recomienda para tratar y mejorar el estado de salud de la persona. Dependiendo de lo identificado en el diagnóstico el nutricionista recomienda una dieta a seguir, a su vez que puede enviar otra serie de exámenes para remitir a otras áreas el tratamiento de enfermedades identificadas y que deben ser tratadas por otros profesionales.

El seguimiento de todo estos pasos ayuda a que el profesional ponga en práctica las habilidades y conocimientos que tiene. De igual forma puede ayudar a detectar enfermedades graves de forma temprana (para que puedan ser tratadas lo más rápido posible) y mejorar la calidad de vida de las personas. Se recomienda que las personas acudan donde los nutricionistas por lo menos una vez al año para así evaluar su estado nutricional, sin embargo, para personas con padecimiento antecedentes que afecten el estado nutricional se recomienda que las consultas sean con menos inérvalo de tiempo, para así tener un mejor control.

#### 2.1.2.3. Simulación

La simulación es una técnica para la práctica y el aprendizaje de conocimientos de una o varias disciplinas. Reemplaza y amplifica experiencias reales con experiencias guiadas que evocan o replican aspectos del mundo real en un ambiente interactivo [Lat10]. Partiendo de lo anterior, se pueden introducir dos tipos de simulación: una que se enfoca en ayudar a desarrollar habilidades técnicas, llamada Simulación de Procedimientos y otra que se enfoca en el desarrollo de habilidades blandas de las personas, llamada Simulación Inmersa. Combinando ambos tipos de simulaciones se obtiene lo que se denomina una Simulación Híbrida [Wik20a], siendo esta la que se creyó que es la mejor opción para usar en el sistema propuesto. Se pensó que esta era la más apropiada por el hecho de que el fin de este proyecto era plantear un software de uso estudiantil y cómo los estudiantes de la Universidad Javeriana se caracterizan por su integridad, un punto clave es poderles brindar apoyo para desarrollar y afinar tanto habilidades blandas como habilidades técnicas.

#### 2.1.2.4. Estrategias de Simulación

La retroalimentación es fundamental para la simulación. Es útil a la hora de implementar experiencias guiadas, las cuales son una parte importante en los sistemas de simulación. Cabe recalcar que la retroalimentación debería ser basada en el proceso de resolución de tareas del estudiante, calidad de las respuestas y tiempo que tomó en resolver las tareas; estos factores son relevantes a la hora de valorar su trabajo. Para este sistema en particular, se creyó necesario el tener dos tipos de retroalimentación [Riv+18]: una que evalúe las respuestas y selecciones del estudiante a medida que va resolviendo una tarea, siendo esta la retroalimentación progresiva y otra que es la retroalimentación final, la cual consiste en evaluar el proceso y las respuestas del estudiante después de completar la tarea.

Un software de simulación será exitoso en la medida que reconozca y use correctamente los niveles de simulación [Lea]. Se han definido cinco niveles de simulación donde progresivamente se le da libertad al usuario de interactuar con la completitud del sistema. Se identificaron los aspectos más útiles en cada nivel y basándose en ellos se diseñaron tres niveles que que se considera ayudaban que el software pudiera cumplir con las expectativas (también se incluyen los respectivos tipos de simulación para cada uno):

- Nivel básico: En este nivel al usuario se le brindaban las secciones de la historia clínica de una forma más fraccionada, esto para que solo se enfocara en una parte a la vez. De igual forma

en cada sección se tenía separado por pantallas cada una de las partes que conformaban dicha sección. Se invitaba al usuario a llenar en campos libres la información que consideraba era la acertada y a medida que se iba seleccionando la opción de guardar, el sistema le brindaba una retroalimentación inmediata (retroalimentación progresiva) sobre los datos ingresados.

- Nivel intermedio: Este nivel fue diseñado con el propósito de brindarle un entrenamiento más avanzado al usuario. El usuario tenía toda las secciones de la HCN, habilitadas para la actividad, en una misma pantalla y podía moverse entre las secciones. Algunos campos, de ser útil, tenían ciertas ayudas o gráficas. De igual forma había una ayuda general por sección en donde se podían tanto texto, como imágenes para brindar mayor información sobre dicha sección. En este nivel se brindaba solo retroalimentación final, después de que el estudiante finalizara toda la actividad.

- Nivel experto: Diseñado para usuarios con previa experiencia en el sistema. Permitía que el usuario navegará a través de todo el sistema con todas las opciones habilitadas y la menor cantidad de ayudas posible. Sólo se brindaran ayudas en las partes que se considere sumamente necesario, de resto se espera que el estudiante pueda llenar correctamente la actividad. Al igual que el nivel anterior, este nivel solo brindaba retroalimentación final, una vez el estudiante había terminado y enviado la actividad a calificación.

#### 2.1.2.5. Arquitectura de software

La IEEE define la arquitectura de software como la organización fundamental de un sistema plasmado en sus componentes, sus relaciones entre sí, y con el medio ambiente, y los principios que guían su diseño y evolución [Eel06]. El estándar se denomina IEEE 1471, pero en 2011 fue reemplazado por ISO / IEC / IEEE 42010: 2011. *La arquitectura de software sirve para que un software pueda proporcionar la capacidad requerida, ser de calidad suficiente, estar disponible cuando se les prometa y ser entregado a un precio aceptable* [Eel06]. La arquitectura hace parte de la segunda etapa del desarrollo de un software, denominada diseño. Se construye a partir de los requisitos obtenidos en la primera etapa y se construye para cumplir los requisitos no funcionales del sistema. Los estilos arquitectónico son modelos conceptuales en los cuales basarse cuando se está construyendo la arquitectura del sistema, son *un conjunto de decisiones de diseño arquitectural que son aplicables en un contexto de desarrollo específico, restringen las decisiones de diseño de un sistema a ese contexto y plantean como objetivo ciertas cualidades para el sistema resultante.* [Eca16].

- Arquitectura en Capas: Este estilo arquitectónico es uno de los más utilizados. Se basa en dividir la aplicación por capas, comúnmente cuatro, para distribuir los roles y responsabilidades entre las diferentes partes.

#### 2.1.2.6. Prototipo Funcional

*Un prototipo funcional es una muestra o modelo de un producto creado para probar un concepto o proceso o para actuar como un accesorio visual para ser replicado, mejorado y aprendido* [3D 15].

En Ingeniería de Software los prototipos cumplen con una funcionalidad limitada. Son utilizados para permitir que el usuario pueda evaluar la propuesta de desarrollo y mostrar como funcionarían ciertos requisitos en el sistema, ayuda a su vez a identificar requisitos que pudieron ser pasados por altos en un inicio pero es necesario ponerlos en el sistema[ING18]. Los prototipos a su vez pueden ayudar a detectar errores más rápido en un software más pequeño y evitar el gasto y tiempo de corregir el software una vez finalizado[Wik20b]. En este proyecto se planteó el desarrollo de un prototipo funcional (basado en la arquitectura del software propuesta para el sistema) que cumpla con los requisitos funcionales y no funcionales propuestos. Este prototipo funcional era un simulador de historias clínicas nutricionales para apoyar el aprendizaje de los estudiantes de nutrición de la universidad Javeriana Cali. El prototipo debía permitir que los estudiantes pudieran resolver historias clínicas nutricionales a partir de casos clínicos brindados por los docentes. A su vez el prototipo debía permitir que el docente le proporcionara retroalimentación al estudiante, y este a su vez debía poder acceder a dicha retroalimentación acorde a la actividad que había llenado.

#### 2.1.2.7. Usabilidad

La usabilidad es el proceso de identificar que tan fácil les resulta a los usuarios utilizar las interfaces diseñadas en el sistema. Por medio de pruebas para testear la usabilidad de un software se pueden identificar las necesidades de los usuarios que garantizan que la aplicación cumpla con sus objetivos de manera eficiente [The17] . Las interfaces del software deben ser sencillas e intuitivas de utilizar, sino los usuarios dejarán de utilizarla. Aunque hay muchos componentes que hacen parte de la usabilidad, Jakob Nielsen [1] los resume en cinco [Nie12]:

- Aprendibilidad: Cuando los usuarios se encuentran con la primera interfaz del software, ¿qué tan fácil es intuir como realizar las tareas básicas de esta?
- Eficiencia: Cuando los usuarios ya están familiarizados con el software, ¿qué tan rápido pueden realizar tareas dentro de este?
- Memorabilidad: Si los usuarios no han usado el software en un tiempo, cuando vuelven ¿Qué tan rápido pueden acordarse del funcionamiento de la página?
- Errores: ¿Cuántos errores cometen los usuarios, ¿qué tan graves son estos errores y con qué facilidad pueden recuperarse de los errores?
- Satisfacción: ¿Qué tanto les gusta usar el software a los usuarios?

#### 2.1.2.8. Mocks

Un mock es utilizado para simular comportamientos reales o esperados dentro del sistema. Para este proyecto de grado se utilizaron dos clases de mocks:

[1] Jakob Nielsen, Ph.D., es Abogado de Usuarios y director del Grupo Nielsen Norman. El Dr. Nielsen estableció el movimiento de “ingeniería de usabilidad de descuento"para mejoras rápidas y baratas de las interfaces de usuario.

- Mockups: Se utilizó la herramienta Figma (una herramienta de generación de prototipos, principalmente basada en la web) para realizar los mocks de las pantallas de la interfaz de usuario. Por medio de esto se buscaba mostrar como se espera que quedase la aplicación real, sin necesidad de la implementación de código para que los usuarios, interesados y demás pudieran verlo. Realizar estos mocks, antes de implementarlos con código en la aplicación real, ayudaban a que si se debían hacer ajustes o comentarios se realizaran sobre el mock y así la implementación final era la esperada al ser la aprobada por las personas que participaban en esa toma de decisiones.
- Mock Server: Se utilizó mockable (una herramienta gratuita para realizar la simulación de llamadas al API o servidor) para simular la comunicación que se esperaba tener con las partes externas de la aplicaciones que no fueron implementadas por este equipo. Con este mock server se espera que cuando se realice a futuro la integración con la parte externa, no sea tan difícil de realizar dado que será pasar de usar endpoints falsos de la aplicación, a los endpoints reales.

### 2.1.3. Trabajos Relacionados

Varias universidades han apostado por la simulación como método de enseñanza en el campo de la Nutrición y la Dietética. Un ejemplo es la Universidad Católica de la Santísima Concepción. La universidad ha optado por el diseño de un simulador de fichas clínicas. El desarrollo consta de una aplicación web que permite a estudiantes y profesores crear historias clínicas y analizar historias ya creadas. Los resultados del proyecto extraídos de la primera prueba piloto fueron satisfactorios, muestran una alta recepción de parte de los estudiantes y promociona el rol activo del estudiante en su proceso de aprendizaje [Tro+18].

El Centro de Ciencia y Salud de Houston, formando parte de La Universidad de Texas, es otro ejemplo. La universidad cuenta con un laboratorio de simulación y evaluación clínica que le permite a los estudiantes aprender y prácticar habilidades en el entorno clínico de la dietética. El laboratorio cuenta con su propio simulador de paciente, un robot llamado Mr. Sims. El robot exhibe condiciones comunes relacionadas con la nutrición para que los estudiantes recolecten datos y creen historias clínicas a partir de ellos. Este laboratorio de simulación ayuda a los estudiantes a aplicar el conocimiento de la Terapia de Nutrición Médica, así como a aumentar su confianza para evaluar a los pacientes, recetar dietas y tratamientos [Cen].

A diferencia de los anteriores softwares diseñados, este proyecto de grado no solo se orientaba en la creación y análisis de historias clínicas nutricionales. La plataforma diseñada también guiaba al estudiante en el proceso de la creación y manejo de estas. Se buscó que el prototipo estuviera nutrido con datos útiles, notificaciones de error, información, ayuda y reconocimiento para darle más apoyo al estudiante al momento de aprender y practicar. Con esto se buscaba brindarle aprendizaje y práctica interactiva virtual a los estudiantes.

Capítulo 3

# Requisitos del sistema

## 3.1. Definición

La IEEE define un requisito como una condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo. Los requisitos describen cómo debe actuar, aparecer o funcionar un sistema [Tha] y la especificación de estos es el primer paso dentro del proceso de desarrollo de un software dado que a partir de ellos se hace el diseño del sistema y el posterior desarrollo, por lo que, si los requisitos quedan mal definidos, el sistema no tendrá la calidad y/o el funcionamiento esperado. Dentro de este proyecto se manejaron dos tipos de requisitos:

- *Requisitos funcionales*: Acorde a la IEEE un requisito funcional es "una función que un sistema o componente debe ser capaz de realiza". Estos definen la base del sistema al expresar cómo el sistema hará lo que debe hacer en términos generales. De igual forma, sirven para especificar el comportamiento del sistema para ciertos casos o entradas que tenga.

- *Requisitos no funcionales*: También llamados atributos de calidad, son los atributos que debe tener el sistema para poder cumplir con los requisitos funcionales. Estos se dan acorde a las necesidades de los usuarios, las limitaciones y presupuesto, entre otros factores [Tha] y dependiendo de lo especificado se afecta la experiencia de usuario.

## 3.2. Especificación de los requisitos

Los requisitos finales del sistema se definieron a partir de reuniones y búsquedas de información. En primera instancia, se asistió a una charla en donde la docente Martha Lucia Lenis, el cliente, explicó cuál era la necesidad de la carrera de nutrición y dietética, y el porqué hasta la fecha no había sido suplida. A su vez, la docente facilitó un formato general sobre el llenado de una HCN en la universidad que se estudió a profundidad para guiar los estándares y métricas a usar en el sistema. Además, se asistió a una clase de la materia de posgrados,*Ingeniería de requerimientos*, en donde los estudiantes realizaron el experimento de empezar a levantar requerimientos del sistema planteado por la docente Lenis. No solo se resolvieron las dudas de cada estudiante, incluyendo al equipo de este proyecto de grado, sino que también salieron a la luz varias ideas y perspectivas interesantes sobre el sistema que no habían sido consideradas. Por último, a partir de la necesidad expresada por la cliente, se realizó una investigación sobre las estrategias de simulación que fomentan el aprendizaje de los estudiantes. A partir de toda esta investigación y de las reuniones realizadas, se pudo concretar la especificación final de los requisitos mostrados a continuación.

### 3.2.1. Requisitos funcionales

Los requisitos se dividieron en cuatro secciones o grandes módulos: requisitos generales, requisitos de una HCN, requisitos de la simulación y requisitos de los niveles de aprendizaje. A manera general, todos los requisitos generales dependen del requisito funcional RGE-001 (Inicio de sesión). A continuación en las figuras 3.1, 3.2, 3.3 y 3.4 se muestran los requisitos funcionales para cada sección.

| Código | Nombre | Descripción | Importancia | Depende de | Estado |
|---|---|---|---|---|---|
| RGE-001 | Iniciar sesión | El sistema debe permitirle al estudiante iniciar sesión con el correo de la Universidad. | Baja | - | Implementado |
| RGE-002 | Visualizar anuncios | El sistema debe mostrarle al estudiante los anuncios dejados por los docentes. | Baja | - | Implementado |
| RGE-003 | Visualizar actividades | El sistema debe mostrarle al estudiante las actividades que puede realizar. | Media | - | Implementado |
| RGE-004 | Visualizar calificaciones | El sistema debe mostrarle al estudiante las actividades que ya realizó y fueron calificadas por el docente. | Media | - | Implementado |
| RGE-005 | Visualizar casos clínicos | El sistema debe mostrarle al estudiante el caso clínico a resolver. | Media | RGE-003 | Implementado |
| RGE-006 | Visualizar historia clínica nutricional | El sistema debe mostrarle al estudiante el formato base de la HCN dado en la actividad. | Alta | RGE-003, RGE-005 | Implementado |
| RGE-007 | Desarrollar la historia clínica nutricional | El sistema debe permitirle al estudiante desarrollar la HCN acorde al nivel de dificultad escogido por el docente. | Alta | RGE-005, RGE-006 | Implementado |

Figura 3.1: Los requisitos generales expresan las funcionalidades generales que tiene el sistema.

| Código | Nombre | Descripción | Importancia | Depende de | Estado |
|---|---|---|---|---|---|
| RHCN-001 | Sección de datos generales | El sistema debe tener una sección para llenar la tarjeta de identificación de una HCN y los datos personales del paciente. | Baja | RGE-007 | Implementado |
| RHCN-002 | Sección de motivo de la consulta | El sistema debe tener una sección para escribir el motivo de la consulta del paciente. | Baja | RGE-007 | Implementado |
| RHCN-003 | Sección de datos antropométricos | El sistema debe tener un sección para llenar los datos antropométricos de una HCN. | Alta | RGE-007 | Implementado |
| RHCN-004 | Sección de datos bioquímicos | El sistema debe tener un sección para llenar los datos bioquímicos de una HCN. | Media | RGE-007 | Implementado |
| RHCN-005 | Sección de datos clínicos | El sistema debe tener un sección para llenar los datos clínicos de una HCN. | Baja | RGE-007 | Fuera del alcance |
| RHCN-006 | Sección de datos dietéticos | El sistema debe tener un sección para llenar los datos dietéticos de una HCN. | Baja | RGE-007 | Fuera del alcance |
| RHCN-007 | Sección de escritura del diagnóstico | El sistema debe tener una sección para escribir el diagnóstico final del paciente. | Alta | RGE-007 | Implementado |
| RHCN-008 | Sección de escritura del plan nutricional | El sistema debe tener una sección para la escritura del plan nutricional recomendado. | Alta | RGE-007 | Implementado |
| RHCN-009 | Notificación de actividad finalizada. | El sistema debe enviarle una notificación al estudiante informando que la actividad se envió de forma correcta. | Baja | RGE-007 | Implementado |

Figura 3.2: Requisitos para llenar una historia clínica nutricional.

| Código | Nombre | Descripción | Importancia | Depende de | Estado |
|---|---|---|---|---|---|
| RSI-001 | Configuración de niveles | El sistema debe estar configurado con niveles de simulación, los cuales van a apoyar al continuo aprendizaje del estudiante. | Alta | - | Implementado |
| RSI-002 | Visualización de ayudas | El sistema debe desplegar una serie de ayudas, con ejemplos, tipos de datos y sugerencias (dependiendo del nivel) para cada módulo. | Alta | RGE-007 | Implementado |
| RSI-002.1 | Visualización de textos flotantes | El sistema debe tener textos flotantes de detalle e información sobre los datos a llenar, en ciertos niveles. | Media | RGE-007 | Implementado |
| RSI-002.2 | Visualización de información general de la sección | El sistema debe tener un botón para mostrar información relevante sobre cada una de las secciones de una HCN. Esta información es la misma en todos los niveles. | Media | RGE-007 | Implementado |
| RSI-003 | Retroalimentación progresiva | El sistema debe darle una retroalimentación progresiva al estudiante, en ciertos niveles establecidos. | Media | RGE-007 | Implementado |
| RSI-004 | Retroalimentación final | El sistema debe darle una retroalimentación final al estudiante, en ciertos niveles establecidos. | Alta | RGE-007 | Implementado |
| RSI-005 | Ofrecer herramientas | El sistema debe ofrecer una serie de herramientas que ayuden al usuario a llenar una HCN. | Baja | RGE-007 | Implementado |

Figura 3.3: Requisitos para la simulación.

| Código | Nombre | Descripción | Importancia | Depende de | Estado |
|---|---|---|---|---|---|
| RNA-001 | Nivel básico | El sistema debe permitir que el docente realice la elección del nivel Básico. | Baja | RSI-001 | Fuera del alcance |
| RNA-002 | Nivel intermedio | El sistema debe permitir que el docente realice la elección del nivel Intermedio. | Alta | RSI-001 | Fuera del alcance |
| RNA-003 | Nivel experto | El sistema debe permitir que el docente realice la elección del nivel Experto. | Media | RSI-001 | Fuera del alcance |
| RNA-004 | Configuración gráficos | El sistema debe permitir que para ciertos datos se pueda anexar la gráfica o tabla establecida para dicho dato. | Media | RSI-001, RNA-001, RNA-002, RNA-003 | Implementado |

Figura 3.4: Requisitos sobre los niveles de aprendizaje de los estudiantes.

A partir de los requisitos funcionales se realizó un diagrama de casos de uso (figura 3.5) que ayuda a ilustrar las acciones que los usuarios pueden realizar dentro del sistema y la relación entre algunas de ellas. Los casos de uso en rojo no hacen parte del alcance de este proyecto. Fueron desarrollados de forma paralela por otro equipo de personas. Como se explicó con anterioridad, este sistema hace parte de un sistema más grande que será integrado a futuro.

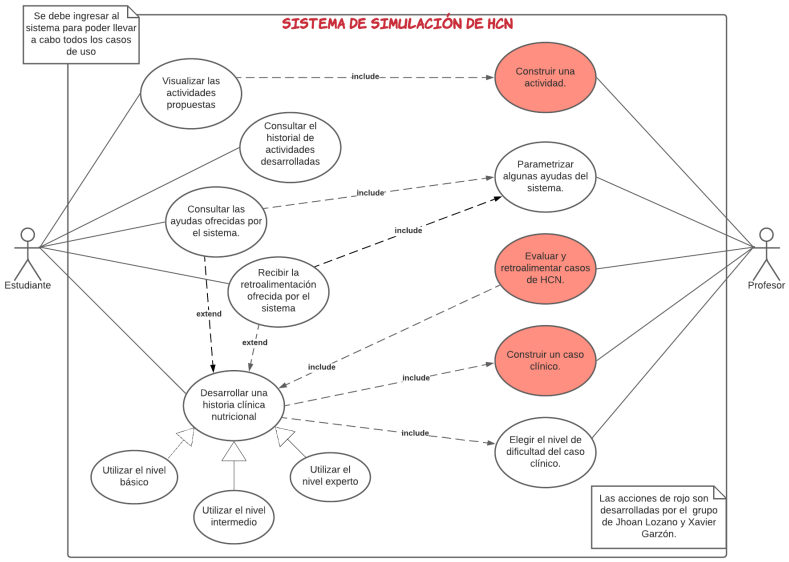



Figura 3.5: Diagrama de casos de uso del sistema.

### 3.2.2. Requisitos no funcionales

En términos generales existen más de 30 requisitos no funcionales que se pueden definir dentro de un sistema. Sin embargo, se deben escoger cuales son los más importantes (acorde a las necesidades del sistema) puesto que la implementación de cada uno conlleva un proceso, además que implementación uno puede significar sacrificar otro. Acorde a las necesidades del cliente y del sistema los requisitos no funcionales (o atributos de calidad) del sistema fueron los que se muestran el figura 3.6. Aún así, en la siguiente sección se explican más a fondo.

| Código | Nombre | Descripción | Importancia | Depende de | Estado |
|---|---|---|---|---|---|
| RNF-001 | Usabilidad | El sistema debe resultar fácil de entender y manejar para el usuario. | - | - | No aplica |
| RNF-002 | Disponibilidad | El sistema debe estar disponible 95% de las veces, durante el período semestral. | - | - | No aplica |
| RNF-003 | Rendimiento | El sistema debe tener tiempos de respuestas de entre 3 y 6 segundos acorde al tamaño de la petición realizada. | - | - | No aplica |
| RNF-004 | Mantenibilidad | El sistema debe poder ser modificable y transferible a terceros (para la continuación de su desarrollo). | - | - | No aplica |
| RNF-005 | Extensibilidad | El sistema debe tener la capacidad de agregarle nuevos módulos o funcionalidades en el futuro. | - | - | No aplica |
| RNF-006 | Escalabilidad | El sistema debe tener la capacidad de manejar aumentos de carga sin disminuir su rendimiento. | - | - | No aplica |
| RNF-007 | Integrabilidad | El sistema debe tener la capacidad de que dos piezas separadas de esta se puedan comunicar correctamente. | - | - | No aplica |
| RNF-008 | Fiabilidad | El sistema debe tener la capacidad de completar acciones que queden pendientes luego de que ocurriera una caída del sistema. | - | - | No aplica |

Figura 3.6: Requisitos no funcionales del sistema.

Capítulo 4

# Diseño y arquitectura

## 4.1. Definición

El doctor Roger Pressman [1] define el diseño de software como un proceso con muchos pasos que se clasifican en uno solo. De forma general, en este paso se establecen las estructuras de datos, la arquitectura general y la representación de interfaces. En otras palabras, se representan visualmente los requisitos definidos en el paso anterior [Cal].

## 4.2. Arquitectura de software

### 4.2.1. Atributos de calidad

En primera instancia, la arquitectura del sistema nace a partir de las necesidades presentadas por los atributos de calidad definidos en la figura 3.6. A excepción del requisito no funcional *Usabilidad*, los atributos de calidad del sistema son:

- Rendimiento: El rendimiento a nivel arquitectónico se entiende como la distribución y tasa de llegada de los servicios y peticiones pedidas, así como los tiempos de respuesta y la latencia del sistema. El rendimiento está estrechamente relacionado con los recursos que cuenta el sistema. Este atributo de calidad es importante en la medida de que se esperan tiempos de respuesta de entre 3 y 6 segundos tanto para las sesiones sincrónicas como asincrónicas.
- Fiabilidad: La fiabilidad se entiende como la capacidad del sistema de poder resolver las operaciones que hayan quedado pendientes con la caída del sistema o el fallo en la prestación de servicios. Este atributo se consideró en la resolución de las actividades con un guardado manual que el estudiante va a poder realizar en todas las partes de la actividad, por lo que si llega a haber alguna falla el estudiante pueda retomar donde guardó por última vez. Estos guardados irán a la base de datos para poder almacenar el progreso de los estudiantes.
- Mantenibilidad: Por mantenibilidad se entiende la facilidad con que se puede modificar, adaptar o transferir el software de un equipo de desarrollo a otro. Este atributo es importante dado que el sistema está pensando para escalar y agregarle nuevos módulos que serán trabajados por otros equipos de desarrollo. Para resolver este atributo se decidió implementar un tipo de arquitectura que será explicado más adelante.

[1] Roger S. Pressman es una autoridad internacionalmente reconocida en el mejoramiento del proceso del software y en las tecnologías de la ingeniería del mismo. Durante casi cuatro décadas ha trabajado como ingeniero de software, gestor, profesor, escritor y consultor, especializado en temas de ingeniería del software

- Escalabilidad: La escalabilidad es la capacidad del sistema para manejar los aumentos de carga sin comprometer el rendimiento de este. Este requisito es petición directa del cliente, además que funcionalidades futuras del sistema van a requerir la capacidad del sistema de escalar con facilidad.
- Integrabilidad: Por integrabilidad se entiende la capacidad de que piezas desarrolladas de forma separada tengan la capacidad de comunicarse y trabajar juntas de forma correcta. Este atributo nace del establecimiento que hay dos grupos separados trabajando módulos independientes que a futuro se integrarán. Para este atributo de calidad se pensó en dejar listos los servicios y endpoints que hacen parte de esa integración y para este trabajo utilizar un mock server que simulara la respuesta que se esperaba de las otras partes del sistema.
- Extensibilidad: La extensabilidad es la capacidad de la arquitectura para manejar la adición de nuevas funcionalidades y componentes. Como se ha mencionado con anterioridad en este sistema se van a implementar otros grandes módulos y funcionalidades por lo que el sistema debe tener la capacidad de manejarlas.
- Disponibilidad: La disponibilidad se entiende como la capacidad del sistema de estar completa o parcialmente disponible cuando se requiera. Este atributo es importante dado que en período semestral se requiere que el sistema esté disponible para la realización de actividades en los cursos en aproximadamente el 95 % de las veces.

### 4.2.2. Diagramas de apoyo

Para la definición técnica de la arquitectura se implementaron algunos diagramas previos para ayudar a estructurar el sistema y su respectiva comunicación. Se realizaron dos diagramas: un organigrama y un diagrama de paquetes.

#### 4.2.2.1. Organigrama

Los organigramas muestran la estructura general del sistema y su división de forma general. A continuación, en la figura 4.1 se puede ver el organigrama definido para este sistema. En la figura se muestra la estructura general del sistema, la cual se divide en tres partes principales: los niveles de aprendizajes (definidos en el capítulo 2), las ayudas que se definirán a continuación y las secciones de una historia clínica nutricional (definida en el capítulo 2).

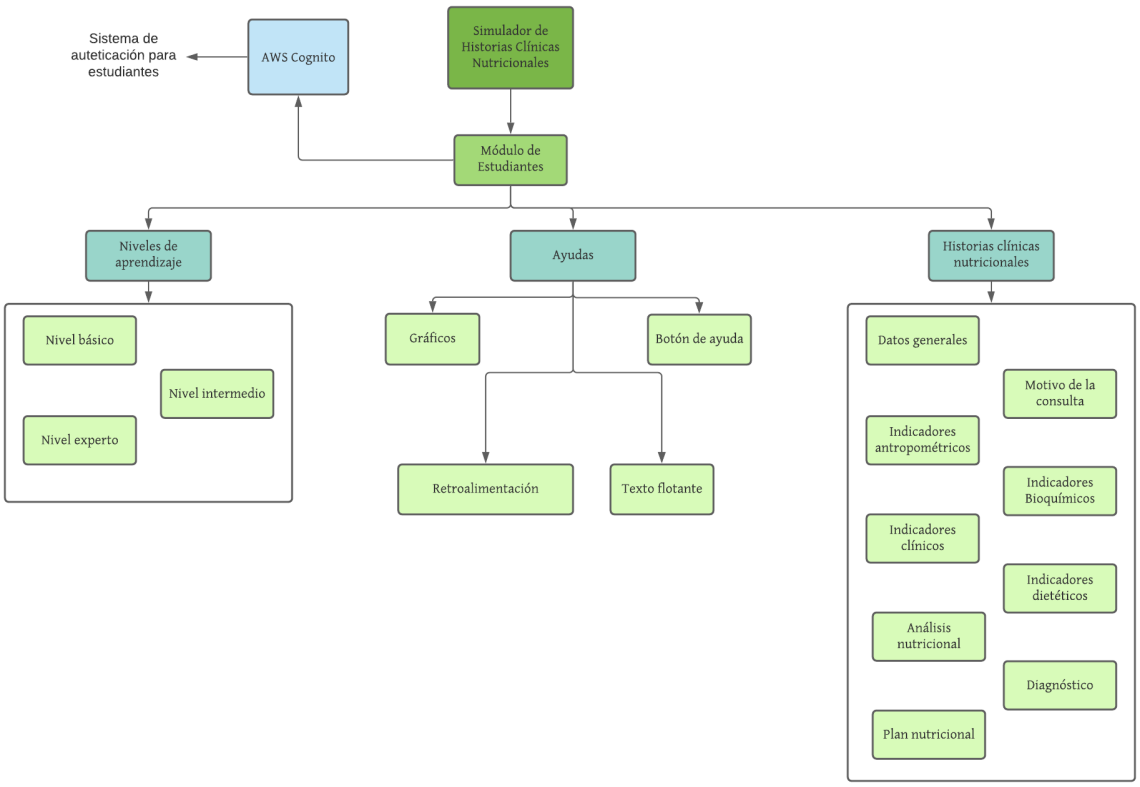



Figura 4.1: Organigrama módulo estudiantes

Al ser este un sistema para el aprendizaje y acorde a lo investigado, se concluyó que el sistema debía proveer ciertas ayudas para garantizar que se cumpla el propósito, es por esto que existen cuatro clases de ayuda que ofrece el sistema.

- Gráficos: Serie de tablas, imágenes, fórmulas o figuras para mostrar ejemplos, información estándar, entre otros.
- Texto flotante: Texto que ayuda a contextualizar al estudiante dándole una escueta definición del campo.
- Botón de ayuda: Brinda apoyo al estudiante ofreciendo ejemplos, amplias descripciones y sugerencias en cada módulo de una HCN.
- Retroalimentación: Como se mencionó con anterioridad se manejarán dos tipos de retroalimentación: una progresiva (análisis de errores cometidos por el estudiante a medida que va llenando campos o seleccionado opciones) y una final (análisis de errores cometidos por el estudiante después de completar alguna tarea o módulo).

Todas estas ayudas buscaban brindar mejores herramientas para el aprendizaje de los estudiantes.

#### 4.2.2.2. Diagrama de paquetes

El propósito de este diagrama es ilustrar las dependencias y relaciones entre los paquetes que conforman cada nivel de abstracción del sistema. Más que documentar el sistema, guía a los desarrolladores a la hora de pensar en el diseño y la implementación de las funcionalidades del mismo. Se muestra el diagrama en la figura 4.2.

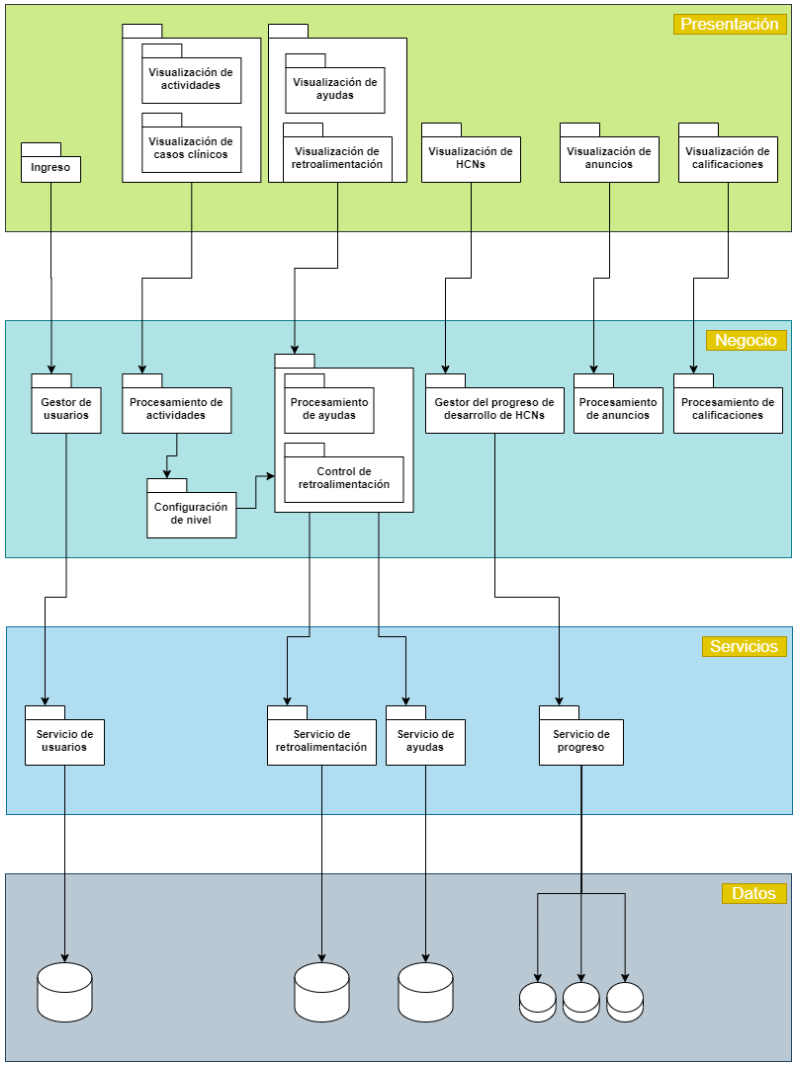



Figura 4.2: Diagrama de paquetes

El diagrama se divide en cuatro capas: presentación, negocio, servicios y datos. Se aprecia en la capa de presentación, la capa más superficial, las vistas y funcionalidades que debe exhibir el sistema al usuario. Algunos paquetes, a su vez, se agrupan en paquetes más generales que abarcan una funcionalidad más grande. Luego cada paquete en esta capa referencia a otro de una capa

menos abstracta, la de negocio. En la capa de negocio se encuentran los paquetes encargados de suplir casi toda la lógica necesaria para poder cumplir con los requisitos del sistema, como por ejemplo, transformaciones u operaciones sobre los datos. Una capa más abajo se encuentra la capa de servicios, esta actúa como un puente entre la lógica de negocio y la información guardada en base de datos. Normalmente, la capa de servicios expone las interfaces que se implementarán en la capa de acceso a datos para la obtención de los datos deseados, incluyendo las restricciones necesarias sobre los datos. Por último, se tiene la capa de datos, su función es representar la necesidad de acceso a los datos desde algún servicio.

### 4.2.3. Modelo C4

*El modelo C4 es un enfoque de "abstracción-primero" para diagramar la arquitectura de software* [C4M]. Se basa en la descomposición estructural del sistema en contenedores y componentes utilizando técnicas como UML (Unified Modelling Language) o ERD (Entity Relation Diagrams). Esto ayuda a que los arquitectos de software y desarrolladores puedan plasmar su pensamiento sobre cómo implementar el sistema y puedan identificar a su vez que patrón o tipo de arquitectura es mejor implementar.

El modelo C4 contiene cuatro niveles de abstracción:

1. Diagrama del contexto del sistema: Ayuda a ver el panorama general del sistema. En este nivel se identifican los usuarios y sistemas externos que interactúan con el sistema a desarrollar. En este punto el detalle no es importante, dado que se enfoca más en los actores y sistemas externos que interactúan entre sí, más que las tecnologías, protocolos y detalles de bajo nivel.

2. Diagrama de contenedores: Ayuda a identificar las diferentes tecnologías y las interacciones entre las diferentes partes del sistema de forma general. Muestra como se reparten las responsabilidades las partes de alto nivel de la arquitectura.

3. Diagrama de componentes: En este paso se descompone cada contenedor para identificar las estructuras que lo componen y como interactúan entre ellos y que tecnología utiliza cada uno para realizar su función o funciones dentro del sistema.

4. Código: En este nivel final se descompone cada componente para mostrar como se implementa por medio de diagramas de clase UML, diagramas de relación de entidades (ERD) entre otros otros.

Para el propósito de este trabajo de grado se implementaron solamente los primeros tres niveles del modelo C4 los cuales se mostrarán a continuación. Cabe resaltar que las tres figuras se encuentran en inglés dado que el desarrollo del software está en inglés (funciones, variables y comentarios) y así ser homogéneo con los nombres dados a los controladores en el código.

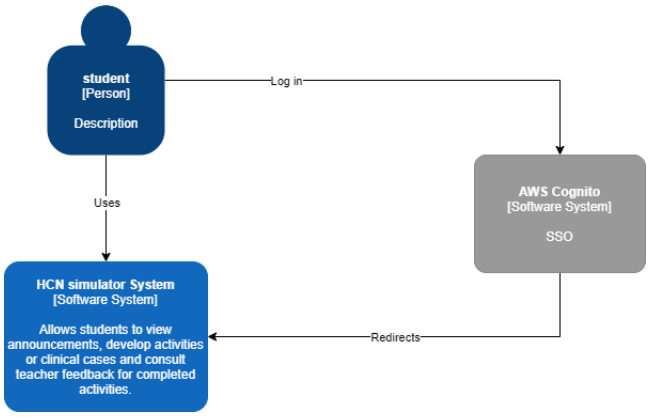



Figura 4.3: Primer nivel: Diagrama del contexto del sistema

En este primer nivel (figura 4.3) se muestra cual es el objetivo principal del sistema. Se tiene el autor, que en este caso es el estudiante, el sistema externo utilizado para la autenticación y el contexto general del objetivo del sistema.

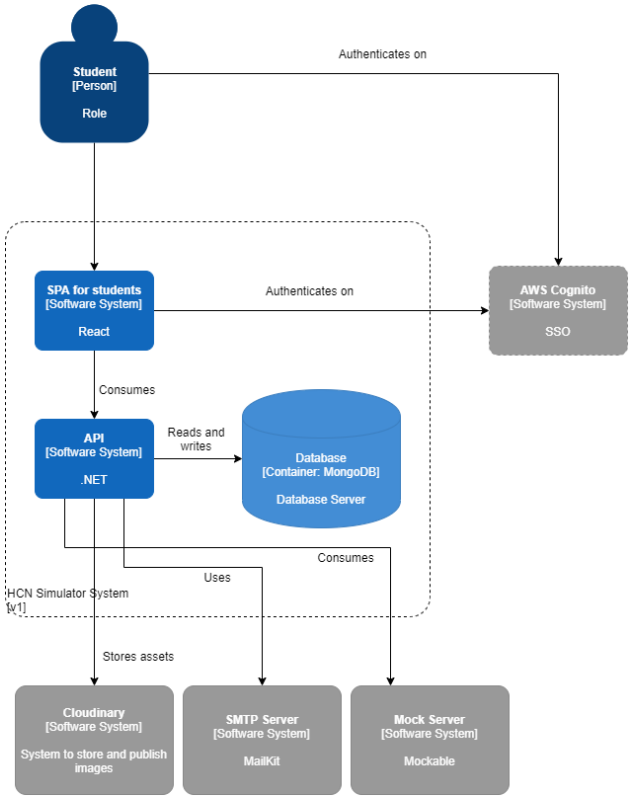



Figura 4.4: Segundo nivel: Diagrama de contenedores

En este segundo nivel, (figura 4.4), se tiene en un contenedor explicado que es lo que hace el

objetivo del sistema pero ya más descompuesto en las tecnologías o sistemas generales, es por eso que se tienen las dos aplicaciones (la UI y el API) y la base de datos que utiliza el sistema. Además de eso se tiene el mismo servicio de autenticación y hay un nuevo sistema externo que se encarga del gestor de imágenes para el aplicación, el cual es Cloudinary.

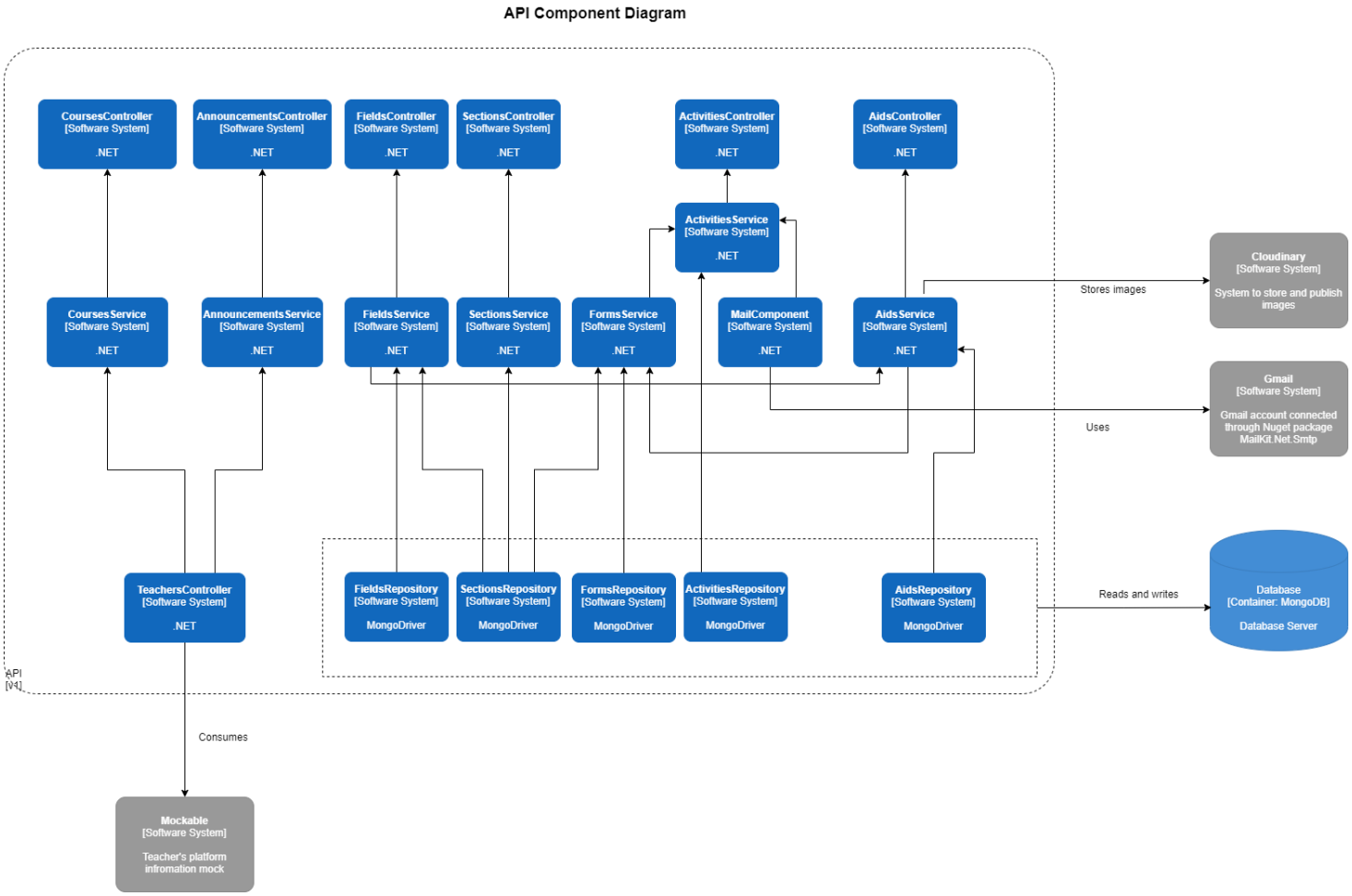



Figura 4.5: Tercer nivel: Diagrama de componentes

En este último nivel, (figura 4.5) se tiene ya descompuesto el contenedor del API a un nivel más técnico. Aunque en el diagrama no se específica con que partes de comunica la UI, esta tiene acceso a todos los controladores por medio de peticiones HTTPS. En este nivel ya se muestra la aplicación más a detalle su implementación y cual es el flujo de comunicación entre los controladores, los servicios y los repositorios.

### 4.2.4. Tipo de arquitectura escogida

Los atributos de calidad que presenta el problema, al igual que el conocimiento de su escalamiento a futuro conllevan a la necesidad de que a futuro se va a necesitar implementar una arquitectura por microservicios para así no comprometer el rendimiento del sistema. Sin embargo, Martin Flower [2], y colegas suyos explican que siempre es mejor empezar con un sistema monolítico, antes que una arquitectura basada en microservicios, aunque se tenga conocimiento que a futuro será necesaria su implementación [Flo15]. Flower explica que la mayoría de historias que ha escuchado o conocido sobre implementaciones exitosas de microservicios vienen de arquitecturas monolíticas que crecieron demasiado y se rompieron, por lo que fue necesario hacer la transición. De igual forma, las historias de implementación fallida de microservicios que conoce se dan por querer empezar de cero con una arquitectura de este tipo.

Aunque se sabe la gran utilidad que tienen las arquitecturas por microservicios, su implementación requiere que los desarrolladores hayan desarrollado previamente arquitecturas que utilizan este patrón, además de conocimiento de automatización de pruebas y despliegue. De igual forma, se requiere la utilización de una gran cantidad de recursos dado que cada microservicio se implementa y despliega de forma independiente por lo que son varias instancias corriendo al mismo tiempo.

Es por esto que, para este proyecto de grado se han decidido implementar dos arquitecturas que estuvieran dentro de los alcances, conocimiento, pero que a su vez siguieran buenas prácticas de programación. Para el desarrollo front-end (o interfaz de usuario) se implementó una arquitectura monolítica, sin embargo no es la tradicional que sigue patrones como MVC (Modelo Vista Controlador), sino una arquitectura conocida como *arquitectura monolítica modular*. El desarrollo backend (o lógica y acceso a los datos) se basa en los principios definidos por la Clean architecure para su implementación.

#### 4.2.4.1. Arquitectura monolítica modular

*Un monolito modular es un enfoque de diseño de software en el que se diseña un monolito con énfasis en módulos intercambiables (y potencialmente reutilizables)* [DEV20]. Esta clase de arquitectura se puede ver como un paso previo antes de pasar de forma definitiva a una arquitectura por microservicios, además de ayudar a solventar algunos de los problemas que tienen las arquitecturas monolíticas y evitar la complejidad que conlleva la implementación de un microservicio. De igual forma, los componentes se pueden estructurar para que, si una parte del sistema está creciendo más que el resto, se pueda coger las partes que lo componen y convertirlo en un microservicio. De esta forma la transición a microservicios va a realizarse de forma más gradual, dado que solo se convertirán en microservicios las partes necesarias.

Como se observa en la figura 4.6, las arquitecturas monolíticas modulares son una tendencia

[2]Martin Flower es un ingeniero de software británico, autor y orador internacional sobre desarrollo de software, especializado en análisis y diseño orientado a objetos, UML, patrones de diseño, y metodologías de desarrollo ágil, incluyendo programación extrema.

que está cogiendo mucha fuerza dado que para muchas compañías el pasar, de forma directa, a una arquitectura por microservicios es imposible por diferentes motivos, por lo que prefieren estructurar el código y los componentes de forma modular hasta que la transición microservicios sea casi obligatoria.

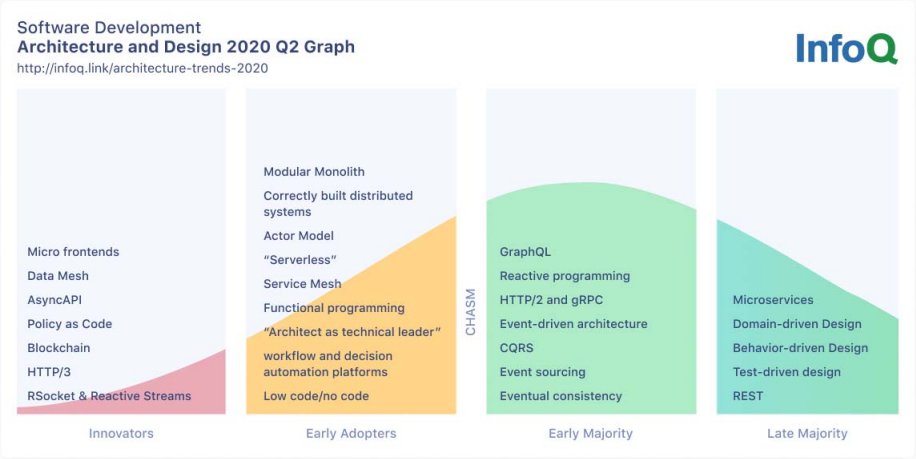



Figura 4.6: Tendencias sobre arquitecturas de software y diseño 2020. Tomado de: https://www.infoq.com/articles/architecture-trends-2020/

### 4.2.5. *Clean architecture*

*Clean architecture* es una implementación de la tradicional arquitectura por capas, pero con un enfoque diferente. En lugar de tener las tradicionales cuatro capas horizontales, *Clean architecture* separa los elementos del diseño en anillos. *Clean architecture* fue creada por Robert C. Martin [3] y *se basa en la premisa de estructurar el código en capas contiguas, es decir, que solo tienen comunicación con las capas que están inmediatamente a sus lados*[Bar]. Basada en el estilo por capas, este tipo de arquitectura separa las responsabilidades en los diferentes niveles y busca ocultar los detalles de la implementación de la lógica de dominio de la aplicación, con esto se consigue una lógica más mantenible y escalable en el tiempo. [Sán16]

La principal característica de esta arquitectura es la regla de dependencia. Esta regla dice que las dependencias de código solo se puede tener de las capas exteriores a las internas. Las capas internas no deben tener conocimiento de las funciones de las capas exteriores y no deben tener referencias de estas. [Sán16].

[3]Robert Cecil Martin, es un ingenierio de software reconocido por desarrollar muchos principios de desarrollo de software y ser un fundador del Manifesto Agil. Cinco de los principios planteados por Martin están dentro de los principios SOLID.

A continuación, en la figura 4.7 se muestra el planteamiento de *Clean architecture*:

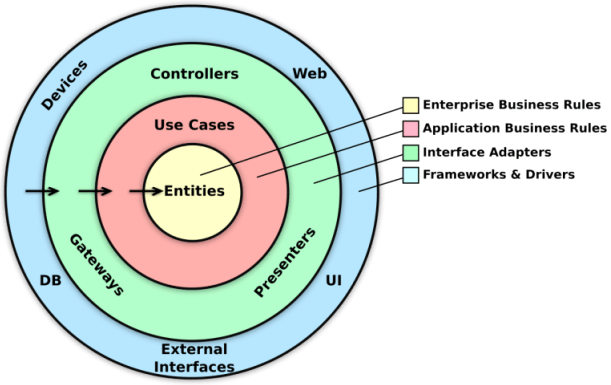



Figura 4.7: Planteamiento Clean architecture

Los anillos que conforman la arquitectura son:

- *External interfaces*: Esta capa no forma parte de la aplicación, son los accesos a los dispositivos externos de la aplicación como la base de datos o la interfaz de usuario.
- *Controllers*: Esta capa define los adaptadores para extraer la información de la capa interna y pasarle dicha información a los servicios externos definidos en la capa de interfaces externas.
- *Use cases*: En esta capa se encuentra toda la lógica de la aplicación.
- *Entities*: Es la capa interna de la aplicación, aquí se define el modelo de negocios, las funciones básicas de la aplicación, entre otros.

### 4.2.6. Arquitectura final

A continuación se presentan dos diagramas. El primero (figura 4.8) representa el diagrama final de la arquitectura del sistema una vez ambas partes de este hayan sido integradas. En este diagrama se muestran las tecnologías utilizadas para el desarrollo de ambas aplicaciones. De igual forma se tiene un Api Gateway, el cual se espera implementar a futuro como una medida de seguridad para proteger los servicios de la aplicaciones de las diferentes interfaces de entrada.

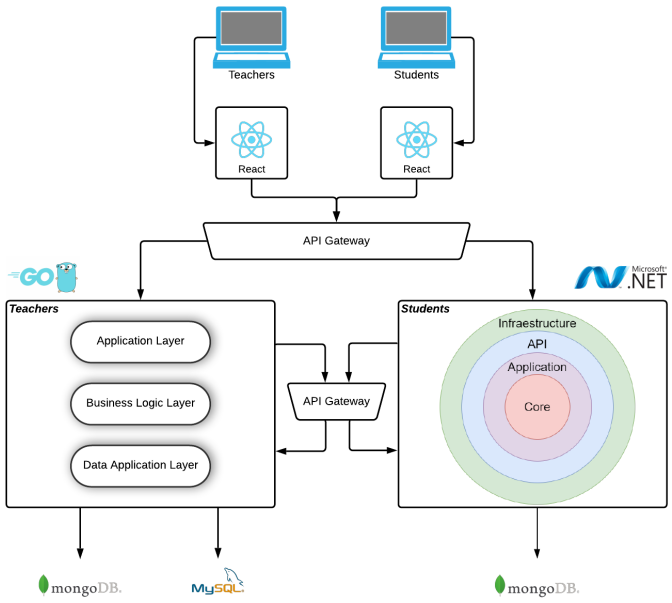



Figura 4.8: Arquitectura de software del sistema

El segundo diagrama (figura 4.9) muestra específicamente la comunicación entre las partes del sistema para este proyecto de grado. Esta arquitectura se divide de cuatro partes, cabe resaltar que en la siguiente sección del documento se explica más a fondo cada tecnología y su respectiva implementación:

- Interfaz de usuario: Como se muestra en la figura, la interfaz de usuario se desarrolló con la herramienta React JS siguiendo el enfoque de arquitectura monolítica modular. Es por esto que todos los módulos del sistema se encuentran almacenados en carpetas, los cuales a su vez de ser necesario también cuentan con sub-carpetas. Por ejemplo, dentro de la carpeta de desarrollo de una actividad(Activity development) se encuentran las carpetas de la interfaz de usuario para cada uno de los niveles explicados anteriormente. La carpeta de Compartido, o shared, son partes de código que se consideran son utilizados por más de una parte del sistema por lo que se decidió guardar en una carpeta diferente para poder ser referenciado de ser necesario por las sub partes.

- Lógica y acceso a datos: El desarrollo de toda la lógica de la aplicación se realizó en .NET siguiendo el enfoque de *Clean architecture* explicado anteriormente. La arquitectura cuenta con cuatro capas de abstracción las cuales son *Core o Entities*, *Application o Use cases*, *API o Controlles* e *Infreastructure o External interfaces*. Cada capa solo tiene dependencias de su capa previa, siguiendo plenamente el enfoque que plantea este tipo de arquitectura.

- Almacenamiento de datos: Como gestor de base de datos se escogió MongoDB, una base de datos no relacional. La razón de esto será explicado en la siguiente sección. En esta parte del

sistema se realiza el guardado de todos los datos relevantes e importantes como las actividades publicadas por los docentes, el progreso de los estudiantes desarrollando dicha actividad y la retroalimentación posterior.

- Almacenamiento de usuarios: Por último, el manejo de los usuarios (tanto su creación como inicio de sesión) se decidió realizar con Amazon Cognito. Al principio se tenía pensando poder llegar a un acuerdo con el Centro de Servicios Informáticos (CSI) de la Universidad, para que así los estudiantes pudieran utilizar sus credenciales para iniciar sesión, sin embargo, dicha integración no se pudo realizar. Por tal motivo se decidió utilizar otra herramienta que tuviera un comportamiento similar al esperado cuando se realice la integración con el sistema de la universidad. Amazon Cognito es una herramienta de AWS que permite gestionar la creación y manejo de usuarios. Más adelante se explicará como se realizó tal integración con este servicio para tener un inicio de sesión y registro completamente funcional.

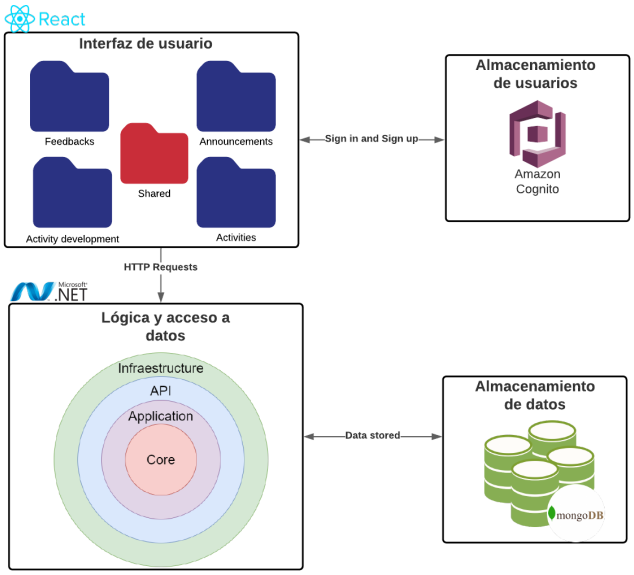



Figura 4.9: Arquitectura de software del proyecto de grado

## 4.3. Diseño de interfaces

En este paso de desarrollo de software también se presentan los primeros mock-ups de interfaces de usuario del sistema. Para esto, se utilizó la herramienta Figma, un editor de gráficos vectoriales y una herramienta de creación de prototipos. Las interfaces se realizaron para que el sistema pudiera ser intuitivo y fácil de manejar para los usuarios siguiendo el objetivo que se busca de usabilidad.

En la figura 4.10 se muestra la vista de las actividades que tiene pendientes por entregar un

estudiante. Cada actividad viene configurada con su fecha de entrega, una descripción corta de lo que trata y el nivel de aprendizaje con el que se va a realizar. Además de eso, se decidió agregar un campo en donde se mostraba el progreso de una actividad, esto para brindar una mejor experiencia de usuario.

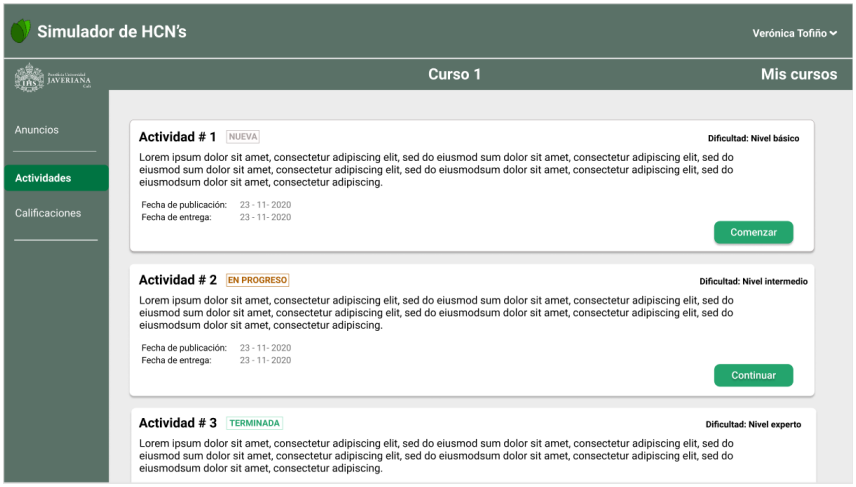



Figura 4.10: Vista de actividades por realizar

En la figura 4.11 se muestra la vista de la ayuda de la sección, una vez el usuario ha seleccionado el botón de información del módulo (definido como botón de ayuda) de color verde. Ahí se muestra información general sobre lo que trata la sección. Al ser una ventana Pop-up con todo el espacio de pantalla disponible, puede incluir información y/o gráficas acorde a lo que se defina es más idóneo mostrar como ayuda de la sección.

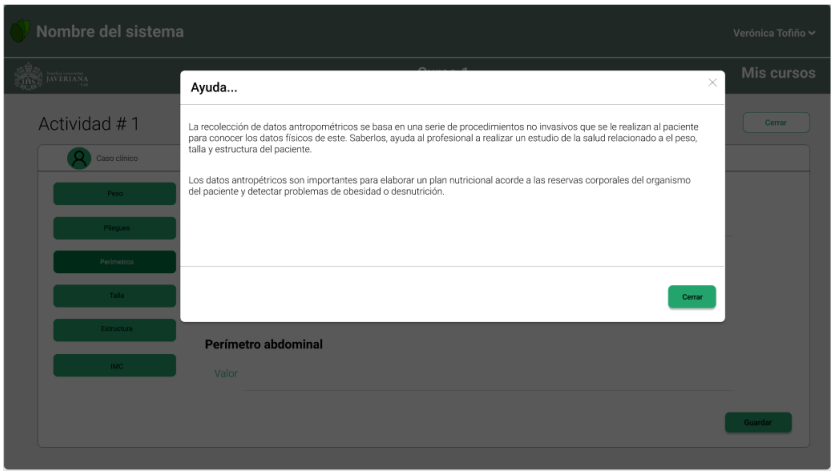



Figura 4.11: Ayuda sobre la sección de la HCN que se está resolviendo

En la figura 4.12 se muestra la vista general de una actividad configurada en el nivel básico. Como se ha mencionado antes el proceso es un poco guiado, por lo que el estudiante no tiene mucha libertad sobre el sistema. Además, se tiene una interfaz más centralizada en la parte de la actividad que se está desarrollando, para que así el estudiante pueda concentrarse en una sección a la vez.


Figura 4.12: Actividad configurada en nivel básico

En la figura 4.13 se muestra la vista general de una actividad configurada en el nivel intermedio. A diferencia del nivel básico mostrado en la figura 4.12, en este nivel el usuario tiene más libertad y se le muestra la información más acorde al formato físico de la HCN trabajado en la universidad Javeriana Cali, se ofrecen ayudas más generales del sistema, como por ejemplo si el usuario se para sobre los campos de la sección y el nivel está configurado para mostrar información, se le desplegará un Tooltip al usuario mostrando información concisa sobre ese campo.

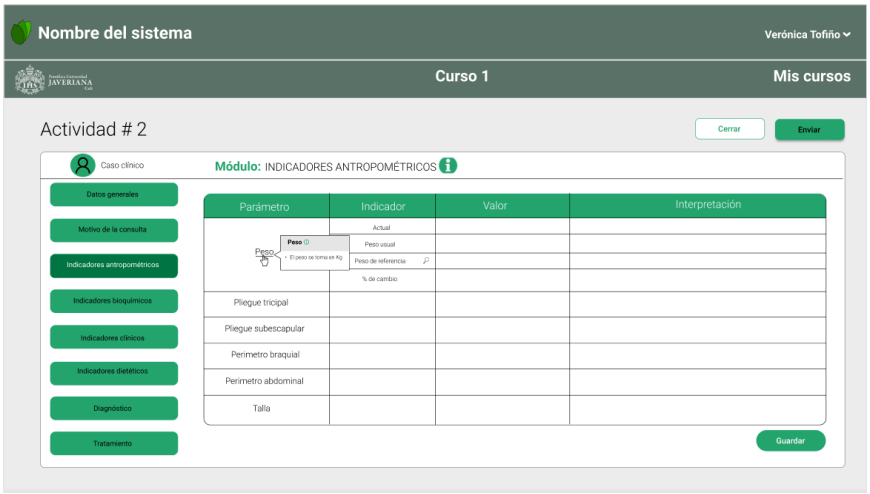



Figura 4.13: Actividad configurada en nivel intermedio

## 4.4. Estructuras de los datos

Después de estudiar a profundidad la clase y cantidad de datos que se iban a manejar para esta aplicación, se decidió implementar un motor de base de datos noSQL enfocado al manejo de documentos. La razón de esto es que gran parte de la información manejada por esta aplicación es sobre la historia clínica nutricional, la cual al final se puede ver como un documento. Para este proyecto se escogió la base de datos MongoDB, dicha tecnología se explica en el siguiente capítulo.

En la figura 4.14 se puede observar el diagrama para la base de datos no relacional:

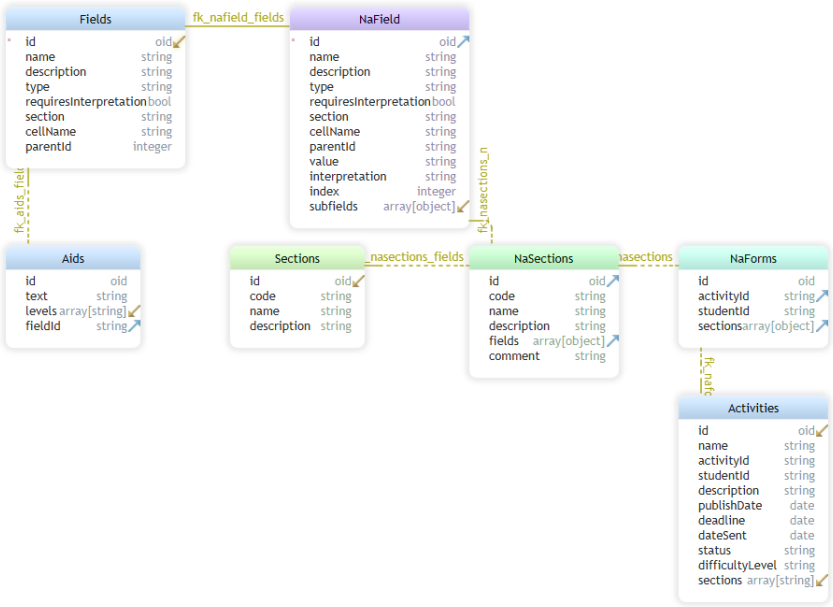



Figura 4.14: Diagrama para la base de datos No relacional

Todo el diagrama está pensado para resolver una historia clínica, es por eso que todas las tablas (a excepción de la tabla de *Activities*) están creadas para darle manejo a la HCN. Las tablas o modelos son:

- *Fields* (campos): Son todas las partes que conforman una historia clínica nutricional. Como por ejemplo Peso, Talla, Exámenes, entre otros. Estos campos cuando se crean están asociados a una sección de la HCN. También tienen un parentId por si son sub-campos (como por ejemplo los indicadores del peso).
- *NaField* (Campos de la HCN): Los campos de la HCN contienen los mismos atributos de los campos (tabla de *fields*), sin embargo cuentan con dos nuevos atributos: *value* e *interpretation*. Estos NaFields se utilizan cuando se está construyendo una nueva historia clínica nutricional

para una actividad dado que el llenado de cada historia clínica es diferente para cada uno de los estudiantes, por lo que van a tener valores diferentes en los attributos *value* e *interpretation*.

- *Sections* (secciones): Las secciones son las partes macro que conforman las historias clínicas nutricionales. Por ejemplo Antropometría, razón de la consulta, tratamiento, entre otros.
- *NaSections* (secciones de la HCN): Al igual que los campos, cuando se está construyendo una nueva historia clínica nutricional se crea una copia de la sección pero en específico para un estudiante. La razón de esto, es porque las secciones tienen una sección de comentarios que es donde el docente escribe la retroalimentación de lo que llenó el estudiante.
- *NaForms* (Formulario de la HCN): Los formularios son la estructuración de las historias clínicas nutricionales. En esta tabla se guarda el progreso de lo que el estudiante escribe en los campos y cuando el profesor retro-alimente sobre lo escrito. Cada formulario tiene el id del estudiante y el id de la actividad general creada por el docente (dado que la actividad en un inicio se crea para todo el curso).
- *Aids* (Ayudas): En esta tabla se guardan todas las ayudas de cada campo. Cada ayuda a su vez tiene los niveles de dificultad para los que se puede mostrar.
- *Activities* (Actividades): Por último las actividades son las que contienen la información sobre lo que se va a desarrollar, cosas como el nivel de dificultad y qué secciones están habilitadas (porque el docente puede deshabilitar secciones de la HCN para una actividad dada). Estas actividades están por estudiante dado que cada estudiante tiene su propio estado de progreso.

Capítulo 5

# Desarrollo

## 5.1. Definición

De acuerdo con IBM Research [IBM], el desarrollo de software se refiere a "un conjunto de actividades informáticas dedicadas al proceso de creación, diseño, implementación y soporte de software". Se resalta también, que abarca todo lo relacionado con el código, pensar, escribir, ejecutar y mantener código. Por esto es importante elegir tecnologías y lenguajes adecuados para la producción del código.

## 5.2. Tecnologías

Las tecnologías que se utilizaron para la implementación del sistema fueron: React para el desarrollo front-end, .NET para el desarrollo back-end; el manejador de versiones se hizo con Github, se utilizó un gestor de base de datos no relacionales llamado MongoDB para el almacenamiento de datos y se decidió realizar la creación y manejo de los usuarios del sistema con una aplicación de AWS llamada Amazon Cognito. Cómo último se decidió utilizar AWS para desplegar la aplicación en la nube.

### 5.2.1. React

React es una librería JavaScript enfocada hacia el desarrollo Front-End y mantenida por Facebook y una comunidad activa de compañías y desarrolladores. Su principal característica es la propuesta de desarrollo de interfaces de usuario por medio de estructuras denominadas componentes. Las razones por las que se eligió esta tecnología para trabajar la parte visual de la aplicación web fueron:

- El desarrollo mediante componentes brinda la posibilidad de crear componentes tan granulares como se necesite e integrarlos entre sí, contribuyendo a que se pueda mantener un código modular, reusable y más fácil de manejar en equipo.
- Otra ventaja es que el framework se presta para la creación de aplicaciones tipo single-page, dando más flexibilidad y libertad en el diseño Front-End.
- Según la curva de aprendizaje de la tecnología, es una buena opción para quienes incursionan en el mundo del desarrollo Front-End, lo que favorece bastante.

- Por último, React ofrece un DOM virtual, lo cual representa una característica interesante para experimentar y así, evaluar la gran ventaja sobre el rendimiento y la experiencia de usuario tan mencionada en las tendencias actuales.

### 5.2.2. .NET

Como tecnología original para implementar el back-end de la aplicación web se tenía Kotlin, sin embargo, después de cierta investigación se encontró que el propósito principal de Kotlin era proveer herramientas para desarrollo móvil más robusto. Con en eso en mente, se decidió buscar una tecnología orientada hacia el desarrollo web y que a la vez esté siendo solicitada en el mercado actual. Como resultado se eligió el framework de .NET, el cual está basado en C#, todo soportado por Microsoft. .NET es una plataforma de desarrollo de código libre y multi-plataforma para crear varios tipos de aplicaciones. Entre las ventajas de usar esta tecnología, no solo están las mencionadas previamente, sino también las siguientes [Mic19]:

- Arquitectura multi-capa: .NET ofrece por defecto una arquitectura por capas que separa el código en responsabilidades como la persistencia y la presentación. Esta estructura permite desarrollar aplicaciones flexibles de manera fácil y en equipo.

- Despliegue y mantenibilidad: el uso de .NET habilita varias herramientas que facilitan el despliegue de la aplicación, incluso permiten hacerlo por partes y luego unirlas cuando sea necesario.

- Soporte: la comunidad de esta tecnología ha crecido exponencialmente, se considera que es una comunidad muy activa y la documentación oficial es descriptiva, fácil de seguir y tiene en cuenta los posibles errores de cada proceso.

- Desarrollo ágil: brinda a los desarrolladores la oportunidad de enfocarse más en el negocio y construir aplicaciones más rápido.

Entre otras ventajas se encuentra la prevención de errores críticos a la hora de programar y la rápida toma de acción ante vulnerabilidades.

### 5.2.3. Github

La elección de esta herramienta de control de versiones se basa en que se necesitaba desarrollar un proyecto en común de manera concurrente y en máquinas separadas. De igual forma, Github es mundialmente reconocido por sus beneficios en la gestión de proyectos de desarrollo de software en equipo. La herramienta brinda la posibilidad de tener varias ramas al mismo tiempo, lo cual permite el trabajo en paralelo de diferentes actividades y su posterior unión mediante la misma herramienta. Principalmente, se escogió la herramienta dado que al ser solo dos desarrolladoras, no hay límites significativos sobre la gestión del repositorio.

### 5.2.4. MongoDB

MongoDB es un sistema de gestión de bases de datos no relacionales, orientado a documentos. El principal motivo de la necesidad de un gestor de bases de datos NoSQL se debió a que cierta parte del sistema necesitaba guardarse en una base de datos enfocada a los documentos pues por su estructura tan compleja, sería poco eficiente estructurarla en una base de datos relacional. MongoDB siendo uno de los motores más conocidos y fáciles de utilizar e integrar a un proyecto, fue el que se decidió usar. Además que permite un gran rendimiento, escalabilidad horizontal de ser necesario y al no utilizar esquemas permite tener una mejor eficiencia.

### 5.2.5. Amazon Cognito

Amazon Cognito es una herramienta desarrollada y mantenida por AWS. Se utiliza para dar manejo a la autenticación, almacenamiento y control de los usuarios dentro de aplicaciones web y móvil desarrolladas. Algunas de las ventajas que provee son [AWS]:

- Directorio de usuarios seguros y escalable: Cuentan con directorios para usuarios de forma segura con capacidad de escalado para lograr el manejo de millones de usuarios al tiempo.
- Autenticación basada en estándares: Este servicio está construido a partir de los estándares de identidades definido, lo cual permite que se pueda realizar la autenticación de usuarios mediante Oauth 2.0, SAML 2.0 y OpenID Connect.
- Seguridad para aplicaciones y usuarios: Amazon Cognito reúne las condiciones establecidas por HIPAA y cumple con los requisitos de las normas PCI DSS, SOC, ISO/IEC 27001, ISO/IEC 27017, ISO/IEC 27018 e ISO 9001.
- Integración sencilla con aplicaciones desarrolladas: Amazon Cognito cuenta con integración en diferentes frameworks de desarrollo web y móvil, lo que permite que se pueda configurar de forma sencilla las funciones para el control de inicio de sesión, suscripción y confirmación de usuarios.

### 5.2.6. AWS

Era necesario implementar una plataforma para desplegar la plataforma de forma pública. Las dos opciones eran a través de un server con ip pública en la red o a través de una plataforma Cloud. Se decidió la opción de una plataforma Cloud, sin embargo la condición era que debía proveer los servicios necesarios bajo la capa gratuita, esto por los recursos del proyecto. Se realizó una investigación entre la nube de Azure, Google Cloud y AWS evaluando lo que ofrecía la capa gratuita de cada una bajo lo que se necesitaba en el proyecto. Al principio se decidió utilizar la nube de Azure, sin embargo después de utilizarla por unos días se decidió que los servicios en la capa gratuita que ofrecían no eran suficientes para lo que necesitaba, de igual forma no era tan intuitiva de utilizar, esto dado que Azure es una herramienta con mucho potencial pero para desarrolladores con previa experiencia en temas de infraestructura y prácticas de DevOps y CI/CD.

Finalmente se decidió utilizar la nube de AWS dado que provee todos los servicios necesarios (AWS Cognito, EC2, IP Elástica para publicar en red) dentro de su capa gratuita para la capacidad de este proyecto de grado y ya se tenía una previa experiencia utilizándola por lo que volverla a usar para desplegar este proyecto fue más sencillo.

### 5.2.7. Docker

Docker es una aplicación que permite empaquetar el software y servirlo en contenedores alojados sobre el sistema operativo de la máquina que se use para desplegar la aplicación, lo que hace que sea una virtualización ligera y basada en el mapeo de puertos. Lo anterior es útil para este proyecto porque el despliegue de software por medio de contenedores brinda más economía a la hora de escalar el software cuando sea necesario, los contenedores son relativamente fáciles de gestionar y son la base para una infraestructura más organizada y menos interdependiente entre ella ya que cada contenedor es independiente de los otros. Para hacer el proceso de despliegue más limpio y eficiente se complementó Docker con el uso de la herramienta Compose para definir múltiples contenedores, cada uno en su propio ambiente aislado, desde un solo archivo. Así, descargar las imágenes necesarias, montar y desmontar los contenedores se puede lograr con unos pocos comandos.

## 5.3. Implementación

A continuación, se explica como se implementó lo definido con anterioridad para el desarrollo de la aplicación.

### 5.3.1. Front-end

La interfaz de usuario fue implementada siguiendo plenamente el enfoque de una arquitectura modular. Es por esto que, los componentes fueron separados en carpetas individuales en su mayoría, buscando así que si un componente cambiaba en su comportamiento no afectara a las demás partes del sistema.

En la figura 5.1 se muestra la separación por carpetas del código, al utilizar un enfoque modular se busca dar una mejor organización de código, además de la posibilidad de que si el sistema crece demasiado sea fácil la extracción de la carpeta necesaria dado que todo se da por medio de referencias.

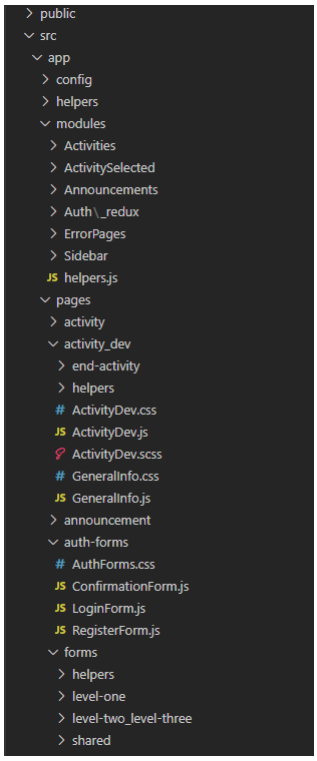



Figura 5.1: Implementación enfoque modular en la interfaz de usuario

De igual forma se utilizó la reutilización de código para una mejor mantenibilidad. En las siguientes figuras (5.39, 5.3 y 5.4) se muestra como se llaman a componentes externos en lugar de definirlos dentro de un mismo código para una reutilización de código.

```
import { Button, Modal } from "react-bootstrap";
import React from "react";
import { Text } from "react-native";

export function ModalInfo({ show, handleClose, textInfo }) {
  return (
    <Modal show={show} onHide={handleClose} size="lg">
      <Modal.Header closeButton>
        <Modal.Title> Ayuda... </Modal.Title>
      </Modal.Header>
      <Modal.Body className="p4">
        <Text>{textInfo}</Text>
      </Modal.Body>
      <Modal.Footer>
        <Button className="button" onClick={() => handleClose()}>
          Cerrar
        </Button>
      </Modal.Footer>
    </Modal>
  );
}
```

Figura 5.2: Definición del componente ModalInfo

En esta primera figura se define el componente. En este caso es el Modal de ayuda que despliegan toda las seccione de la historia clínica cuando se selecciona el botón de ayuda. Como todas las secciones utilizan el mismo modal, lo que cambia es la información que contiene, se decidió crear un componente para el modal que tuviera como parámetros de entrada el booleano de si debe mostrarse el modal o no, la acción para cerrar el modal y por último la información que debe mostrar dicho modal cuando se llame.

```
import { TitleSection } from "./helpers/TitleSection";
import { ModalInfo } from "./GeneralInfo";
import { toAbsoluteUrl } from "../../helpers";
import { AnthroForm } from "../forms/level-two_level-three/anthropometry/AnthropometryForm";
import { ToastContainer, toast } from "react-toastify";
```

Figura 5.3: Llamado de componentes externos

En esta siguiente figura se muestra una porción de código en donde se hace el llamado del Modal definido en el otro archivo. Este llamado se hace a través de un Import, seguido del nombre del componente, y luego la ruta del archivo dentro de las carpetas de la aplicación.

```
272   <div className="widget-dev">
273     {TitleSection(name)}
274     <div className="grid">
275       <div id="upleft">
276         <Fas icon={faUserCircle} className="regularIcon" />
277         <button
278           href="#"
279           style={{
280             background: "transparent",
281             border: "none",
282           }}
283         >
284           Caso clínico
285         </button>
286       </div>
287
288       <div id="upright">
289         <span style={{ color: "#00000", fontWeight: "bold" }}>Módulo: </span>
290         {titleModule}
291       </div>
292       <div className="break" />
293       <hr className="breakLineDev" />
294       <div className="d-flex">
295 >       <div className="buttonGroup">···
317
318 >       <div className="Form" id="myDiv">···
326         {showModalInfoA && <ModalInfo show={showModalInfoA} handleClose={() => setShowModalInfoA(false)} textInfo={textInfoA} />}
327         {showModalInfoG && <ModalInfo show={showModalInfoG} handleClose={() => setShowModalInfoG(false)} textInfo={textInfoG} />}
328         {showModalInfoCR && <ModalInfo show={showModalInfoCR} handleClose={() => setShowModalInfoG(false)} textInfo={textInfoCR} />}
329         {showModalInfoD && <ModalInfo show={showModalInfoD} handleClose={() => setShowModalInfoG(false)} textInfo={textInfoD} />}
330         {showModalInfoT && <ModalInfo show={showModalInfoT} handleClose={() => setShowModalInfoG(false)} textInfo={textInfoT} />}
331         {showModalInfoB && <ModalInfo show={showModalInfoB} handleClose={() => setShowModalInfoB(false)} textInfo={textInfoB} />}
332       </div>
```

Figura 5.4: Uso de los componente Modal

En esta última figura se muestra otra porción de código donde se utiliza el componente Modal. Como se puede ver de la línea 326 a la línea 331, el llamado del modal se realiza para cada una de las secciones cuando se selecciona el botón de ayuda.

Con la reutilización de código se pudo abstraer la lógica del Modal en un archivo, con esto se busca darle una mayor mantenibilidad e independencia al código.

#### 5.3.1.1. Redux

Redux es una librería Javascript de código abierto para el manejo de estados de las aplicaciones. Es muy utilizado por librerías como React o Angular para manejar el tema de estados e información que se pasa de un componente a otro.

El principal motivo de utilizar Redux, fue por la necesidad que tenía el proyecto de tener que pasar información de unos componentes a otros, además de que a partir de cierta información que pasa de un sitio a otro, los componentes del sistema deben mostrar cierta información.

Aunque se sabía que se podía utilizar la memoria del Browser para realizar el paso de estados y para guardar información necesaria, como los datos de sesión, se decidió que personas con conocimientos básicos de computación podían acceder a esta memoria y ver o cambiar datos que no se quería que se cambiaran. Con eso en mente se decidió implementar redux dentro del sistema para agregar una capa más de protección a la aplicación.

En la sección de anexos (Anexo #1) se explica de forma más detallada la implementación de Redux.

#### 5.3.1.2. Conexión con AWS Cognito

Como se explicó con anterioridad, la autenticación de usuarios se realizó con AWS Cognito. Esta herramienta ofrece un plugin el cual puede ser instalado en aplicaciones React para así generar una conexión segura a través de los endpoints de Cognito y poder gestionar el manejo de los usuarios. El plugin instalado fue **aws-amplify**, el cual permite hacer la conexión con AWS y particularmente con cognito de forma sencilla y rápida.

Esta implementación permite que, si a futuro se realiza la integración con el sistema de la universidad, el cual se espera tenga un comportamiento similar, solo se deberían definir nuevas credenciales de seguridad y cambiar los endpoints para el manejo de información, dado que la aplicación ya estaría en capaz de interpretar la nueva información de los servicios de la universidad, así como hace en este momento con AWS.

En el anexo #2, se explica como se generó la conexión con Cognito para el manejo de los usuarios.

#### 5.3.1.3. ESLint y prettier

Para tener un código más limpio, se decidió utilizar dentro del proyecto ESLint y Prettier. Ambas herramientas son utilizadas por los desarrolladores de código para el análisis de código estático de archivos Javascript.

A continuación, en la figura 5.5 se muestra el archivo de ESLint con las reglas definidas para el análisis de los archivos.


Figura 5.5: Configuración de ESLint para el proyecto.

Para este proyecto se definió que las reglas a utilizar fueran las recomendadas para aplicaciones React para manejar el estándar definido.

### 5.3.2. Back-end

Para empezar la estructuración del backend, como ya se explicó, se basa en una arquitectura monolítica donde toda la lógica se centraliza en un solo API. El orden de carpetas se inspiró en el template CleanArchitecture de https://github.com/jasontaylordev/CleanArchitecture/tree/main/src y se adaptó la estructura a las necesidades del proyecto y el estilo que se pensó más conveniente. Cada carpeta es, de hecho, un proyecto independiente que se podría integrar en otras soluciones como una librería, incluso se podría aislar cada proyecto por separado y cada uno podría componer su propia solución si se quisiera extender la arquitectura a un enfoque más distribuido. La organización de las carpetas se puede observar en la figura 5.6

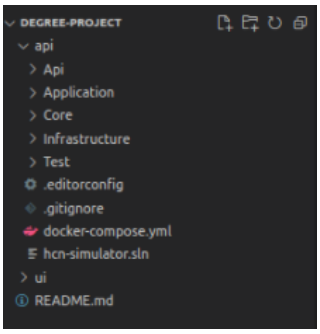



Figura 5.6: Estructura general de carpetas.

A continuación se explica la naturaleza y el uso de cada carpeta. La solución *hcn-simulator.sln* está compuesta por:

**Api**: Es un proyecto construido usando la plantilla ASP.NET Core Web App. Como se observa en la figura 5.7, está compuesto a su vez por una carpeta de *Controllers* la cual alberga la colección de controladores que ofrece el API para dar manejo a las peticiones HTTP que son enviadas al back-end. Luego se tiene la carpeta de *Handlers*, la cual se añadió para separar las responsabilidades de manera más visual. Este tipo de carpetas usualmente ofrecen funcionalidades variadas, aquí particularmente sólo alberga un manejador de excepciones que sirve como *middleware* para *catchear* o atrapar cualquier excepción que arroje el sistema. Después se tiene la carpeta de *Mappers* en la cual se hizo uso de la librería AutoMapper para declarar los perfiles necesarios para traducir los *DTOs* (Data Transfer Objects) que son enviados al API como datos de entrada, a los modelos de negocio con lo que trabaja el back-end como tal.

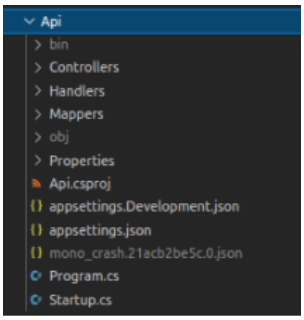



Figura 5.7: Estructura del proyecto Api.

**Application**: Este proyecto fue construido usando la plantilla de Class Library que ofrece el dotnet CLI. El propósito de esta capa de aplicación es realizar toda la lógica de negocio necesaria. Bajo un modelo purista de *Clean Architecture* esta capa sólo debería depender de la capa de Dominio y solo contener interfaces de los servicios para luego implementarlas en la capa de infraestructura. Se tomó un acercamiento distinto a este modelo y se separaron las interfaces y sus implementaciones según los modelos de negocio en los que se enfocan, como se ilustra en la figura 5.8 en el caso del modelo *Activity*. En esta capa del back-end es donde se generan la mayoría de las excepciones cuando se incumple alguna regla de negocio.

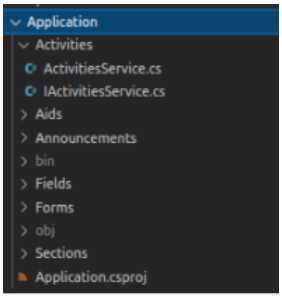



Figura 5.8: Estructura del proyecto Application.

**Core**: Esta capa también es referida como capa de dominio y se implementó usando la plantilla de Class Library. Contiene la definición de todas las entidades, modelos, excepciones y tipos referentes al negocio. De esta capa dependen todas las demás capas. Como se observa en la figura 5.9 contiene las declaraciones de los *DTOs* que el API puede recibir y enviar, las excepciones personalizadas que se pueden devolver al cliente y por último, los modelos, también agrupados por responsabilidades de negocio.

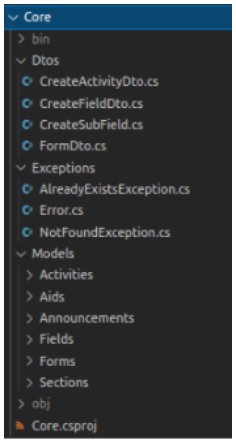



Figura 5.9: Estructura del proyecto Core.

**Infrastructure**: Esta capa permite que el API pueda realizar conexiones externas a ella y conectarse con aplicaciones *third-party*. El ejemplo más común es la conexión con el servidor de base datos. Como se ve en la figura 5.10 en este caso particular todo lo relacionado a la persistencia se apartó en la carpeta de *Persistence*, además se tiene un apartado *TeacherApp* que se refiere a la conexión con el *mock* de datos que proveería el sistema de HCN usado por los docentes. También se tiene un controlador de correos que permite notificar al estudiante cuando se envió una actividad con éxito y está lista para ser calificada por el profesor de la asignatura.

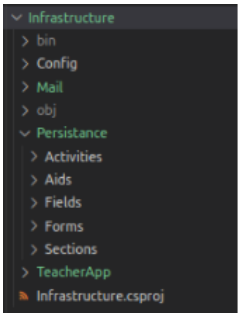



Figura 5.10: Estructura del proyecto Infrastructure.

**Tests**: Por último se tiene un proyecto creado a partir de la plantilla Class Library y cuyo objetivo es tener la colección de pruebas sobre el código back-end para asegurar el debido funcionamiento y

algunos atributos de calidad del API.

### 5.3.3. Patrón Repositorio/Servicio y DI

Para lograr el grado de flexibilidad y mantenibilidad que se espera del sistema, se implementó el patrón Repositorio/Servicio que consiste en manejar los datos, la parte menos abstracta de todo el sistema, en repositorios los cuales son inyectados en los servicios para que estos puedan realizar toda las operaciones y lógica de negocio que se necesite sobre una o varias colecciones de datos. El patrón establece que cada repositorio se encarga de un solo tipo de dato y así se tienen repositorios para cada entidad que se maneje en el sistema. Mientras que los servicios se encargan de transformar los datos que pueden extraer de uno o varios repositorios de acuerdo a la necesidad que se tenga. Este patrón se basa en la inyección de dependencias (DI, por sus siglas en inglés), para funcionar correctamente

En el proyecto se decidió manejar repositorios separados para todas las entidades del sistema. Particularmente algunos repositorios acoplados entre ellos, por ejemplo, los repositorios de las entidades que se refieren a los campos y las secciones del documento de la historia clínica nutricional, más específicamente, los repositorios de las clases *Field* y *NaField* (Nutritional Assessment Field) y, *Section* y *NaSection* (Nutritional Assessment Field). Dado que el núcleo del proyecto es la historia clínica nutricional, la clase *Field* es una clase que viene a ser secundaria y su función principal es ayudar a construir la estructura del documento, referido en el código como un objeto tipo *NaForm*, el cual es fuertemente anidado, puesto que se compone de Nutritional Assessment Sections, y estas a su vez, de Nutrional Assessment Fields. La relación entre las clases referentes a campos se observa en la figura 5.11. De manera semejante se utiliza la clase *Section* en el caso de las secciones. Luego, la relación entre las clases referentes a las secciones se aprecia en la figura 5.12.

```
24 references
public class NaField : Field
{
    [BsonElement("value")]
    2 references
    public string Value { get; set; }

    [BsonElement("interpretation")]
    2 references
    public string Interpretation { get; set; }

    [BsonElement("index")]
    2 references
    public int Index { get; set; }

    [BsonElement("subfields")]
    5 references
    public IList<NaField> Subfields { get; set;}

    0 references
    public NaField()
    {
    }
```

Figura 5.11: Clase NaField

```
23 references
public class NaSection : Section
{
    [BsonElement("fields")]
    6 references
    public IList<NaField> Fields { get; set;}


    [BsonElement("comment")]
    1 reference
    public string Comment { get; set;}

    0 references
    public NaSection()
    {
    }
```

Figura 5.12: Clase NaSection

Finalmente, en la figura 5.13 se ilustra como se relaciona el documento de la historia clínica nutricional con ambas entidades (campos y secciones). Por la anterior razón, se encontró válido el acople entre las colecciones de datos tipo *Field* y *NaField* y para las colecciones de tipo *Section* y *NaSection*, aunque son tipos distintos, no puede existir ningún campo en el documento que no haya sido creado primero entonces y solo se partieron en repositorios distintos para respetar el patrón de diseño, de manera semejante se organizan las secciones. Cabe resaltar que esta estructura de clases permite manipular de manera más general el documento de la historia clínica nutricional porque no se necesita una clase para cada campo, ni para cada sección y también brinda la modularidad suficiente como para incluir las secciones que el docente desee en el documento de historia clínica de cada actividad propuesta.

```
29 references
public class NaForm
{
  [BsonId]
  [BsonRepresentation(BsonType.ObjectId)]
  4 references
  public string Id { get; set; }


  [BsonElement("activityId")]
  2 references
  public string ActivityId { get; set; }


  [BsonElement("studentId")]
  1 reference
  public string StudentId { get; set; }

  [BsonElement("sections")]
  4 references
  public IList<NaSection> Sections { get; set;}

  2 references
  public NaForm()
  {
  }
```

Figura 5.13: Clase NaForm.

Otro aspecto fundamental para aprovechar los beneficios que trae la Clean Architecture es hacer uso de la Inyección de Dependencias (ID). El concepto viene de las bases SOLID, más concretamente del principio de Inversión de Dependencias (Dependency Inversion), depender sobre interfaces y no sobre las implementaciones concretas. La inyección de interfaces permite desacoplar el código, de manera que solo se debe cambiar una pequeña porción para elegir usar una implementación en concreto sobre otra. La utilidad de lo anterior en este sistema radica, por ejemplo, en los servicios a los que se les inyectan interfaces de repositorios, así, si se cambia de un repositorio implementado para una base de datos Mongo a un repositorio en memoria, ninguna clase que use dicho repositorio sufriría cambios.

En la figura 5.14 se muestra la interfaz para el repositorio de ayudas para cada campo de la historia clínica nutricional y en las figuras 5.15 y 5.16 se muestran dos implementaciones diferentes de la interfaz. La elección de qué clase implementar en el código, para .NET se realiza en la clase *Startup.cs* donde bastaría con añadir la línea que muestra la figura 5.17.

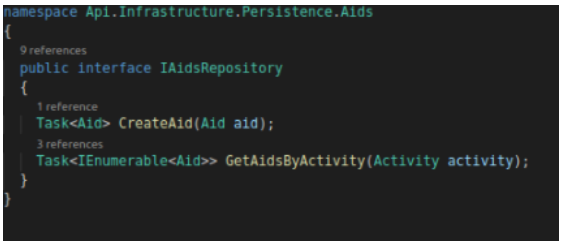



Figura 5.14: Interfaz del repositorio para las ayudas de los campos.


Figura 5.15: Implementación para el repositorio de ayudas en memoria.


Figura 5.16: Implementación para el repositorio de ayudas en MongoDb.

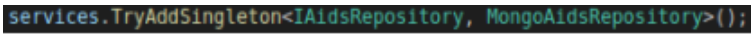



Figura 5.17: Creación de *Singleton* para el repositorio elegido.

### 5.3.4. Prácticas para documentar el código

Las prácticas de documentación son esenciales para asegurar un código mantenible. Por esta razón en el proyecto se procuró comentar las secciones o líneas de código cuya lógica no es trivial. Se utilizaron herramientas propias del lenguaje como la etiqueta *<summary>* para hacer explícito el significado de cada método para contribuir a la facilidad de su uso, se puede apreciar en la figura 5.18. Por último también se utilizó la librería de Swagger para ilustrar qué métodos ofrece el API, el propósito y los parámetros de cada *endpoint*. En la figura 5.19 se hace más explícita la interfaz

de la herramienta en el proyecto. Esta poderosa herramienta fue fácil de instalar y es muy fácil de mantener puesto que analiza el código de manera estática e infiere los controladores, sus métodos y sus tipos con solo algunas anotaciones. Usando Swagger, el proceso de discusión de modelos se vuelve más eficiente y cómodo para los integrantes del equipo de desarrollo.

```
/// <summary>
/// This method supports addition of first degree fields over specified section.
/// </summary>
1 reference
public async Task<NaField> AddFieldToSection(NaField field)
{
  await _naSectionsRepository.AddFieldToSection(field);
  return field;
}
```

Figura 5.18: Uso de etiquetas para documentar métodos.

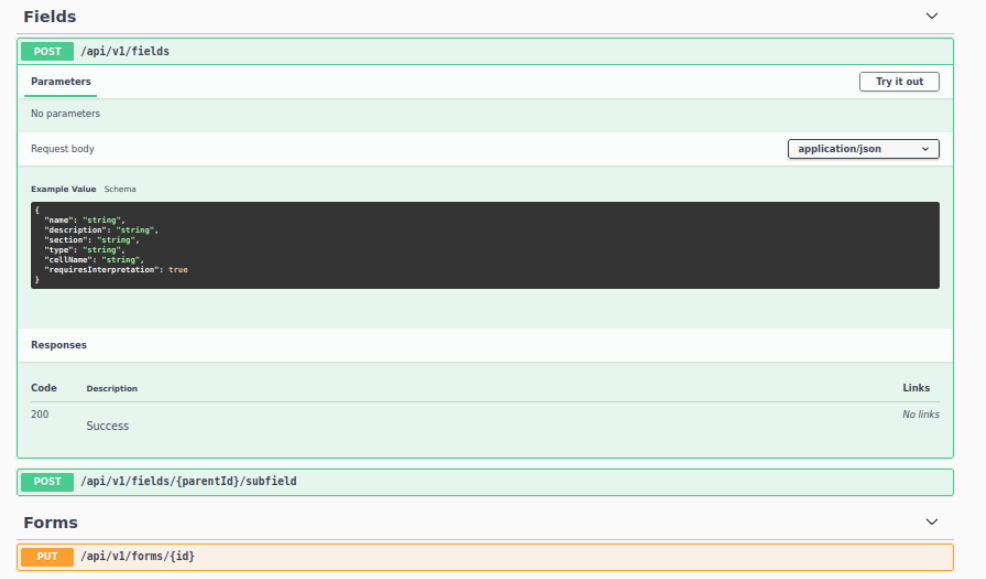



Figura 5.19: Ejemplo de endpoints que expone el API

#### 5.3.4.1. Envío de notificaciones

El requisito del sistema *RHCN-009* era que, cuando el estudiante finalizara de realizar la actividad y la enviara para que el docente pudiera evaluar, si el proceso de envío se realizaba correctamente al estudiante debía llegarle una notificación de correo en donde se dijera que se había enviado con éxito la actividad.

Para implementar este requisito se decidió utilizar el plugin MailKit, una librería de código abierto de .NET la cual soporta protocolos IMAP, POP3, SMTP, entre otros.

La implementación en el código fue relativamente sencilla, se creó en la capa de infraestructura el controlador *Mail* (esto dado que al ser una conexión con un servicio externo, siguiendo los principios de la arquitectura Clean, se debía definir en esta capa), el cual realiza el envío de correo. En la figura 5.20 se puede ver la implementación del controlador:

```
namespace Infrastructure.Persistance.Mail
{
 using Api.Core.Models.Activities;
 using MailKit.Net.Smtp;
 using Microsoft.AspNetCore.Mvc;
 using Microsoft.Extensions.Logging;
 using MimeKit;
 using System.Globalization;
 using System;

 6 references
 public class MailController : ControllerBase
 {
  private readonly ILogger<MailController> _logger;
  0 references
  public MailController(ILogger<MailController> logger)
  {
   _logger = logger;
  }

  1 reference
  public IActionResult SendEmailConfirmation(EndActivity endActivity, string activityName)
  {
   MimeMessage message = new();

   //Configure the sender
   MailboxAddress from = new("Simulador HCN bot",
   "simhcnbot@gmail.com");
   message.From.Add(from);

   //Configure the recipient data
   MailboxAddress to = new(endActivity.RecipientName,
   endActivity.Email);
   message.To.Add(to);

   message.Subject = "¡Su actividad ha sido enviada con éxito!";

   //Construct the body message
   BodyBuilder bodyBuilder = new();
   bodyBuilder.HtmlBody ="<div><h4>¡Su actividad ha sido enviada con éxito!</h4>"+
     "<p>Una vez haya sido revisada por su docente podrá mirar los comentarios seleccionado la actividad en la sección 'Retroalimentación' de la página. </p>" +
     "<p>Igualmente, todas las actividades enviadas que aún no han sido revisadas por el docente, se encuentran en la misma sección para que pueda visualizar cuando cambia de estado. </p>" +
     "<b>Detalles de la entrega:</b>" +
     $"<ul><li><i> Nombre de la actividad: </i>{activityName} </li><li><i> Fecha de envío: </i>{DateTime.Now.ToString("dddd, dd MMMM yyyy hh:mm:ss tt", new CultureInfo("es-ES"))}</li></ul></div>";

   message.Body = bodyBuilder.ToMessageBody();

   try {
    SmtpClient client = new();
    client.Connect("smtp.gmail.com", 465, true);
    client.Authenticate("[redacted]", "[redacted]");
    client.Send(message);
    client.Disconnect(true);
    client.Dispose();
    return new StatusCodeResult((int)System.Net.HttpStatusCode.OK);
   } catch (Exception ex)
   {
    _logger.LogError(ex.ToString());
    return new StatusCodeResult((int)System.Net.HttpStatusCode.Conflict);
   }
  }
 }
}
```

Figura 5.20: Implementación envío de correos con SMTPClient/MailKit.

Esta notificación llega a la cuenta del usuario con el que se inscribió a la aplicación. En la figura 5.21 se puede observar un ejemplo de notificación cuando el estudiante finaliza y envía la actividad.

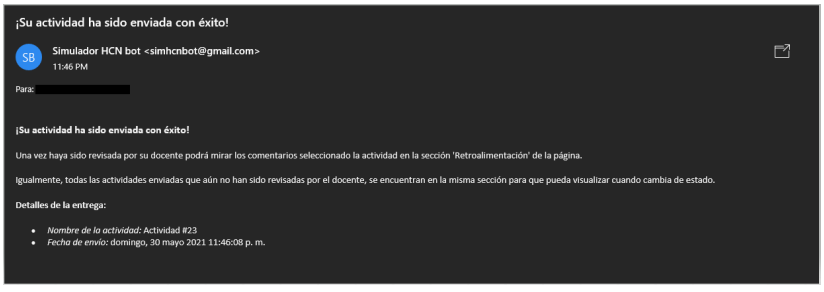



Figura 5.21: Ejemplo notificación de correo

### 5.3.5. Github

Como se explicó en la sección anterior, se eligió Github como la herramienta de control de versiones, esto por el conocimiento y uso previo que se ha tenido con la herramienta.

A continuación, en la figura 5.22 se muestra la vista general de proyecto:

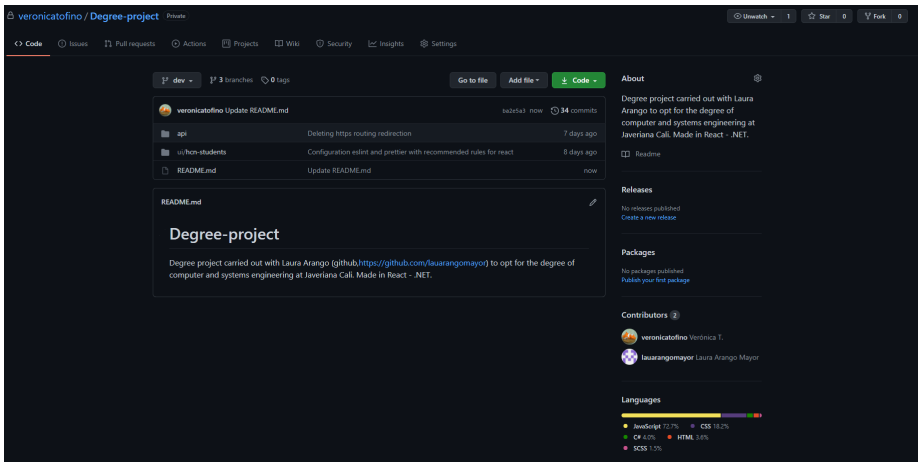



Figura 5.22: Vista general del proyecto en Github

Dentro del proyecto se encuentran, en carpetas separadas, el desarrollo de las aplicaciones back-end y front-end del sistema.

#### 5.3.5.1. Ramas

El proyecto tiene dos ramas principales **dev** y **main**. La rama dev se utiliza para subir todo el desarrollo, una vez ha sido aprobado por otros integrantes del equipo (en este caso al ser solo dos personas obligatoriamente debe ser aprobado por la otra persona), y en la rama main se ponen las versiones estables del producto. Así se busca que si se encuentra algún error al agregar nuevas características a la rama dev, la rama de producto no se vea afectada. Para poder agregar nuevo código a dev, primero se debe seguir un proceso en donde se crea una rama a partir de la rama de dev, en la que se agrega todo el nuevo código, luego de eso se realiza un Peer review (explicado más abajo) y una vez se aprueba el nuevo código se agrega a la rama dev.

Se decidió crear un estándar para el llamado de las ramas (a excepción de dev y main), dicho estándar es: ***feature/short-explication-task/SH-number***. Como se observa en el estándar todo se maneja con lower case y al final, separado por un slash, está el número de la tarea en JIRA.

Todo lo explicado se puede observar en la figura 5.23, en donde se muestran diferentes ramas creadas:

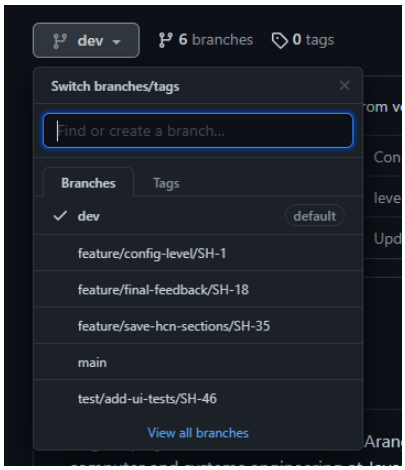



Figura 5.23: Ramas creadas en el proyecto

Como se muestra en la figura anterior, se tiene definidas seis ramas. Las dos principales mencionadas anteriormente y cuatro nuevas ramas. Tres de esas ramas son de funcionalidades nuevas del sistema por lo que llevan el nombre *feature* al inicio. La otra corresponde a pruebas implementadas para el sistema por lo que lleva el nombre *test* al inicio. De esta forma se puede trabajar de forma independiente y paralela en cambios que se vayan a agregar.

#### 5.3.5.2. Pull requests y Peer review

Para lograr proteger las ramas dev y main de que se les agreguen porciones de código sin la revisión y aprobación de otra persona, Github dentro de las configuraciones del proyecto permite se decidió proteger las ramas dev y main con ciertas reglas.

En la figura 5.24 se muestra las reglas configuradas para la protección de la rama dev. se decidió proteger la rama con dos cosas: primero no se puede adicionar código a esta rama de forma directa, primero se debe crear un *pull request* y este debe tener al menos una aprobación para poder adicionar el código a la rama de desarrollo y segundo se requieren que los chequeos de estado deben pasar para poder hacer *merge*, al no tener un CI / CD implementado el único chequeo que se hace es que lo que se vaya a adicionar no tenga conflicto con lo que hay actualmente. Por otro lado, en la figura 5.25 se puede ver como se protege la rama con esas reglas configuradas, por ejemplo si la rama actual está desactualizada con respecto a dev y agregar esas nuevas funcionalidades causa conflictos con el código ya aceptado.

Como se muestra en la figura 5.25, aunque el código ya fue aprobado por otra persona, GitHub no permite que se integre el nuevo código dado que causa conflicto con la existente. La persona entonces debe resolver primeros los conflictos para así garantizar que el código que ya está siga funcionando y que el nuevo pueda funcionar también.

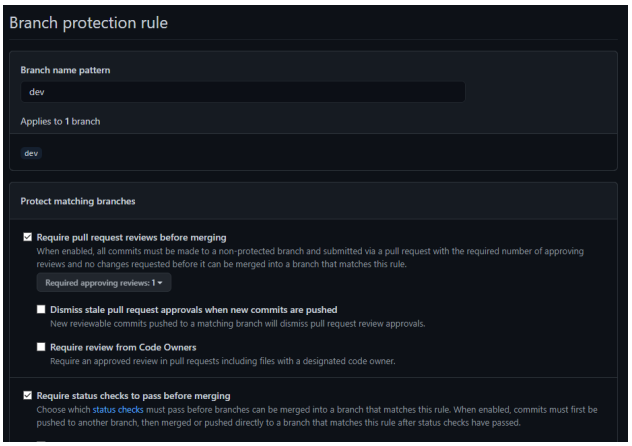



Figura 5.24: Reglas para la rama dev

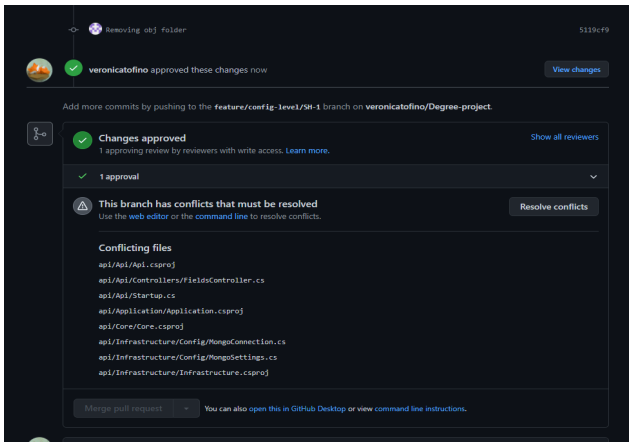



Figura 5.25: Protección de la rama

Al tener las ramas principales protegidas, la única forma de agregar nuevas porciones del código es por medio del enfoque de *peer review*, este enfoque lo que dice es que las nuevas características que se busquen agregar al sistema, primero deben ser aprobadas por otra persona, con los mismos conocimientos y habilidades que la persona programando, asegurando así la calidad del código. Es por esto que antes de agregar características, mejoras o arreglos al código se debe contar con la aprobación de otra persona. Para lograr esto se hace uso de la sección de *pull requests* (PR, de ahora en adelante) de Github. Una vez el PR ha sido aprobado por la otra persona se puede hacer *merge* en la rama dev.

Con todo este enfoque de revisión y protección se buscó garantizar el que se entregara código que pudiera ser fácilmente entendido por otras personas. De igual forma, se buscaba mostrar la implementación primero a otra persona, para que si ella tenía algún comentario mejora o retroalimentación, se pudiese discutir y poder entregar un mejor producto al final.

#### 5.3.5.3. Wiki

Se decidió implementar una wiki dentro del proyecto para los futuros desarrolladores. Con esto se buscaba brindar ciertos tips o ayudas bases para el entendimiento de este proyecto.

En la figura 5.26 se muestra la página de inicio en donde se muestra el menú de lo que hay hasta el momento en la wiki. Por otro lado, en la figura 5.27 se muestra una sección de la Wiki para realizar la instalación del proyecto del API y de la UI.

El objetivo era que a medida que fuese surgiendo información que se consideraba relevante para incluir en la wiki, se fuera actualizando.

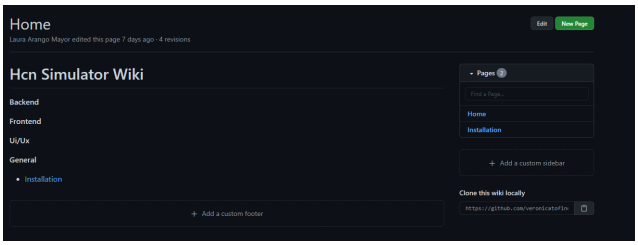



Figura 5.26: Página principal de la wiki del proyecto

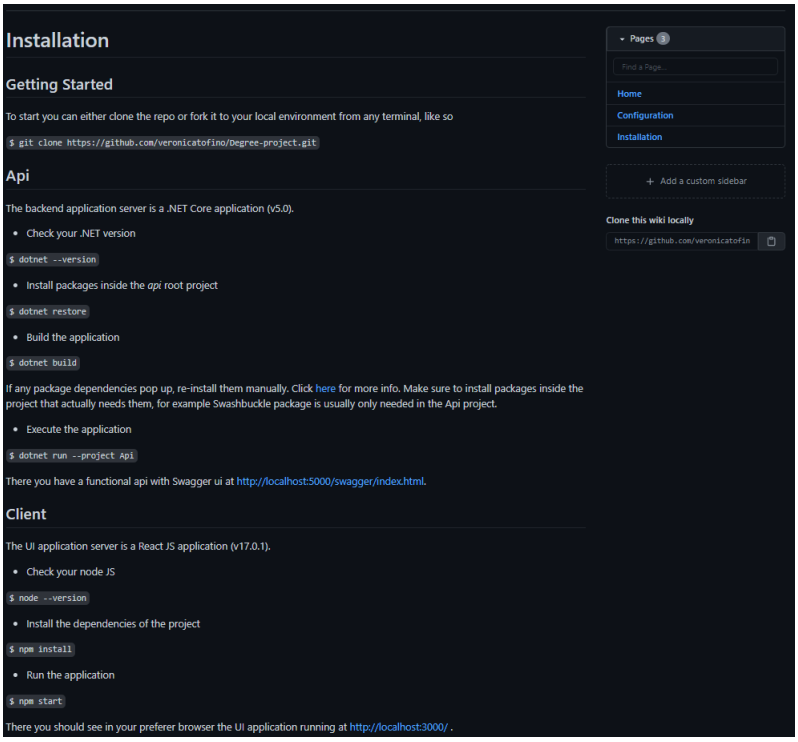



Figura 5.27: Página de instalación de los proyectos

### 5.3.6. SCRUM

Se decidió utilizar la metodología SCRUM para la realización del proyecto, para esto se decidió utilizar la herramienta JIRA para el manejo de los *sprints*. Jira está diseñado para que todos los miembros del equipo de software puedan planificar, supervisar y publicar software de gran calidad.

#### 5.3.6.1. Backlog

En la sección del *backlog* se definían las tareas y subtareas pendientes por realizar por el equipo de desarrollo. Para este proyecto, en el *backlog* se definieron todas las tareas, basadas en los requisitos

funcionales del proyecto, y se les asignó una prioridad, acorde al criterio de los desarrolladores.

A continuación, en la figura 5.28 se muestra el *backlog* del proyecto con los requisitos funcionales escritos en forma de tareas priorizadas:

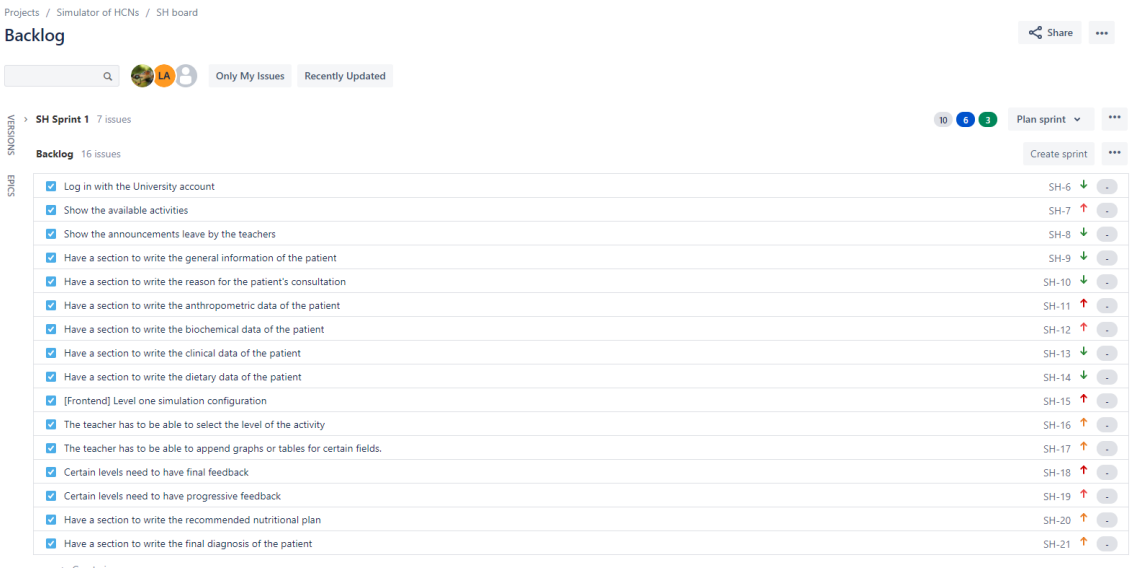



Figura 5.28: Backlog del proyecto

#### 5.3.6.2. Puntos de historia

*Los puntos de la historia o* Story Points *(SPs) son una unidad de medida para expresar o estimar el esfuerzo general que sería necesario para fabricar un elemento de un producto o cualquier otra parte de la producción. Un SP es una medida arbitraria utilizado por equipos Scrum. Esto se utiliza para medir el esfuerzo requerido para implementar una historia (fabricación de un elemento)* [Gue16].

Para este proyecto, se definieron los SPs siguiendo tres criterios:

- Cantidad de trabajo (Amount of work): En esta sección se evalúa que tanto trabajo requiere lo que se quiere implementar.
- Riesgo (Risk): Aquí se mira que tanto riesgo genera para la aplicación lo nuevo que se quiere agregar a la aplicación.
- Complejidad (Complexity): Por último, también se evalúa que tan difícil de implementar es.

Teniendo en cuenta esos criterios los SPs quedaron definidos de la siguiente forma:

| Story points | Amount of work | Risk | Complexity |
|---|---|---|---|
| 1 | Minium changes in the app (CSS, Back) | Don´t have any | Don't have any |
| 2 | Small changes without risk | Don´t have any | Minimum |
| 3 | Small Front or Back changes | Minimum | Minimum |
| 5 | Implementation of componentes or pages, or new features | Small | Medium |
| 8 | Do refactors, reuse old code or changes of the code, small changes in the CI/CD | Medium | High |
| 13 | Big changes in the CI/CD | High | High |

Figura 5.29: Puntos de historia para el desarrollo de las tareas

Como se ve en la figura anterior, un SP significa que la cantidad de trabajo es mínima y no presenta ningún riesgo o complejidad. Dos SPs presentan pequeños cambios, con una complejidad mínima y no presentan ningún riesgo, tres puntos son para pequeños cambios en la UI o en el API que si pueden presentar un pequeño riesgo. Para cinco puntos se definió que la cantidad de trabajo era implementar nuevas funcionalidades del sistema, lo cual presentaban riesgos pequeños y complejidad media al ser cosas nuevas. Por último ocho o 13 puntos presentan una complejidad muy alta, sin embargo ocho puntos son pequeños cambios en el CI mientras que para 13 puntos los cambios son grandes por lo que tiene un mayor riesgo.

Al momento de estimar el tiempo y dificultad de implementar una tarea se utilizaba la de definición de SPs definida por el equipo.

#### 5.3.6.3. Sprint

Los sprint tuvieron una duración de dos semanas. Antes de empezar cada sprint, el equipo escogía las tareas que iban a ser desarrolladas durante esas dos semanas y se realizaba un refinamiento (explicado más adelante) para definir los puntos de historia de cada una de las tareas a desarrollar. Luego de eso, se empezaba el sprint. En un principio, las tareas no estaban asignadas a nadie, esto porque se consideró que en el equipo se contaba con las mismas habilidades, por parte de ambas desarrolladoras, por lo que a medida que una tarea era finalizada la desarrolladora decidía con que tarea deseaba seguir.

A continuación se muestra el tablero de un sprint con tareas en él (figura 5.31) y las SPs asignadas para dichas tareas (figura 5.30).


Figura 5.30: Sprint con los puntos de historia asignados

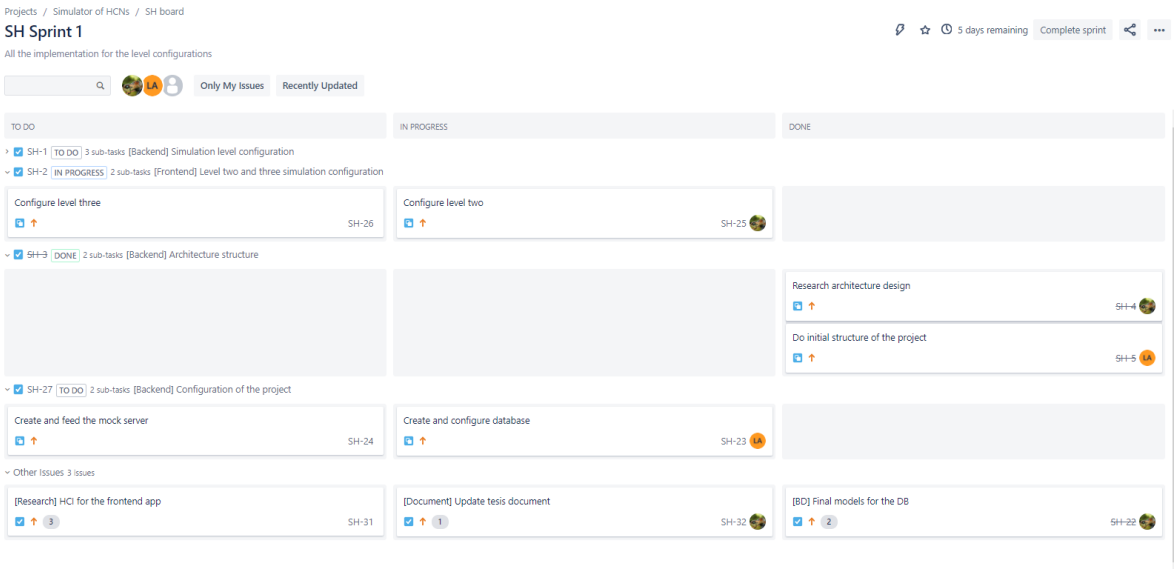



Figura 5.31: Tablero de un sprint

#### 5.3.6.4. SCRUM Master

Para este proyecto de grado, se decidió implementar el rol del *SCRUM Master* para supervisar el desarrollo de los sprints. Principalmente el SCRUM Master cumplía con el rol de supervisar el desarrollo del sprint, y antes de agregar sub-tareas dentro del sprint se debía discutir con el SCRUM Master. El rol del SCRUM Master se iba rotando entre ambas integrantes del equipo, eso quiere decir que cada dos semanas el rol pasaba a otra persona. Esto permitió que se pudiera distribuir la carga cada sprint.

#### 5.3.6.5. Refinamiento

El refinamiento de las tareas que se van a trabajar en el sprint se realiza un día antes del inicio del sprint. Con esta actividad se estimaba en puntos de historia la dificultad de las tareas para que así se pudiera conocer qué tan fáciles o difíciles iban a ser de implementar durante del sprint. Con este ejercicio también se buscaba que los participantes dieran su punto de vista sobre lo que implicaba el desarrollo de la tarea y por ende los puntos de historia que estimaban, esto ayudó a que se pudieran esclarecer dudas o plantear cosas que no se tenían en cuenta y así llegar a un acuerdo sobre el puntaje en conjunto de la tarea.

En las figuras 5.32 y 5.33 se muestra un ejemplo del refinamiento de una tarea, luego de que se llegara a un acuerdo sobre la cantidad de puntos, en el JIRA en la sección de la tarea llamado *Story Points* se ponía el número escogido en conjunto.

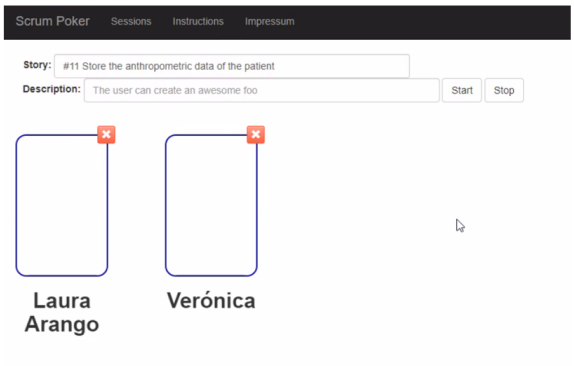



Figura 5.32: Título de la tarea a refinar y nombre de los participantes.

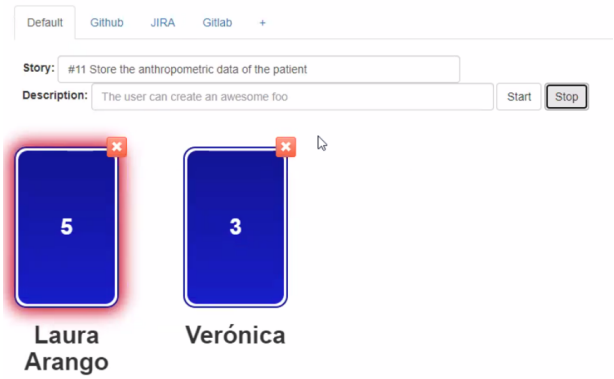



Figura 5.33: Estimación de cada persona de la dificultad en SP.

## 5.4. Mejoras de UI

### 5.4.1. Mejoras de UX

Se consideró necesario mejorar la experiencia de usuario (abreviado UX, en inglés) dentro de la aplicación. Para esto se decidió dividir el trabajo en dos tareas en JIRA.

- La primera tarea consistía en realizar una investigación sobre como mejorar la interfaz de usuario de la aplicación en cuanto a factores de usabilidad, colores, entre otros. Dicha investigación se centró en tres partes principales de la aplicación:
  - Colores: Los colores de una aplicación es una de las cosas más importantes cuando se está desarrollando una aplicación. Es por esto que se decidió realizar una investigación sobre el número, y correcto uso de colores que una aplicación debería tener.

- Botones: Gran parte del desplazamiento por la aplicación se da por medio de botones, es por esto que se consideró necesario ver maneras en que dichos botones podían ser mejorados por medio de aplicación de estilos o tips.
- Tooltips: Al ser una aplicación educativa, muchas ayudas son brindadas al usuario cuando está desarrollando una actividad. Es por eso que se investigó cuales son las cosas a tener en cuenta cuando se implementan botones dentro de la aplicaciones.

Toda la información encontrada relevante se escribió como comentario dentro de la tarea creada en JIRA para realizar dicha investigación y a partir escoger iban a ser las partes a implementar.

A continuación en las figuras 5.34, 5.35 y 5.36 se muestran los resultados encontrados para cada una de las secciones:

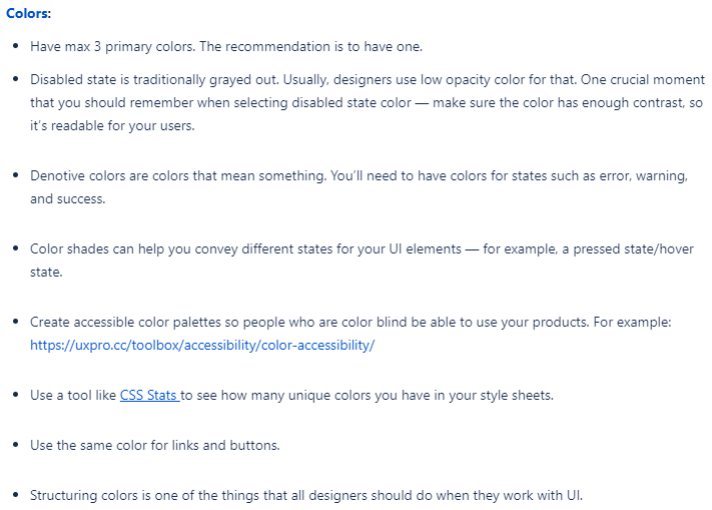



Figura 5.34: Información recolectada sobre uso de colores.

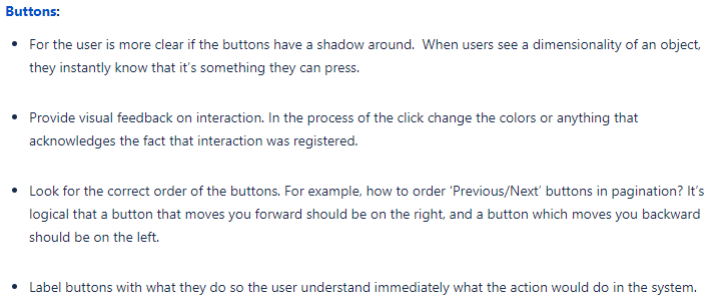



Figura 5.35: Información recolectada sobre uso de botones.

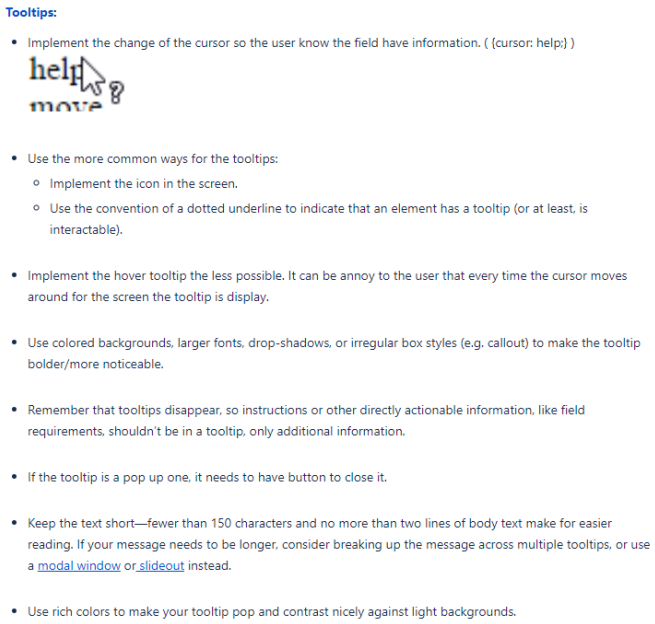



Figura 5.36: Información recolectada sobre uso de tooltips.

- La segunda tarea fue implementar las mejores escogidas de todo lo investigado. Esta tarea tenía a su vez dos sub-tareas. La primera era implementar las mejores a manera de mockups para así evaluar como se veían antes de implementarla como código. Una vez aprobadas, se pasaba a implementar en la aplicación real dichos cambios.

  En la tarea de JIRA se definieron cuales iban a ser las mejoras escogidas para implementar, las cuales fueron:

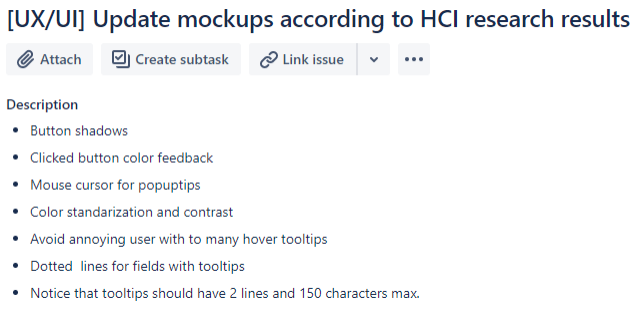



Figura 5.37: Descripción de las mejoras a implementar.

  - Implementación y discusión de los mockups: Se implementó una propuesta, a excepción de la implementación de colores donde se plantearon varias, para la implementación de cada ítem escrito en la tarea, después de eso se expuso el resultado y se tomó la decisión

sobre cuales iban a ser las implementaciones finales en el sistema. Específicamente para los colores se plantearon entre tres y cuatro combinaciones de colores para la aplicación. En la figura 5.37 se pueden ver diferentes propuestas de colores y en la figura 5.39 se puede observar las propuestas finales aceptadas.

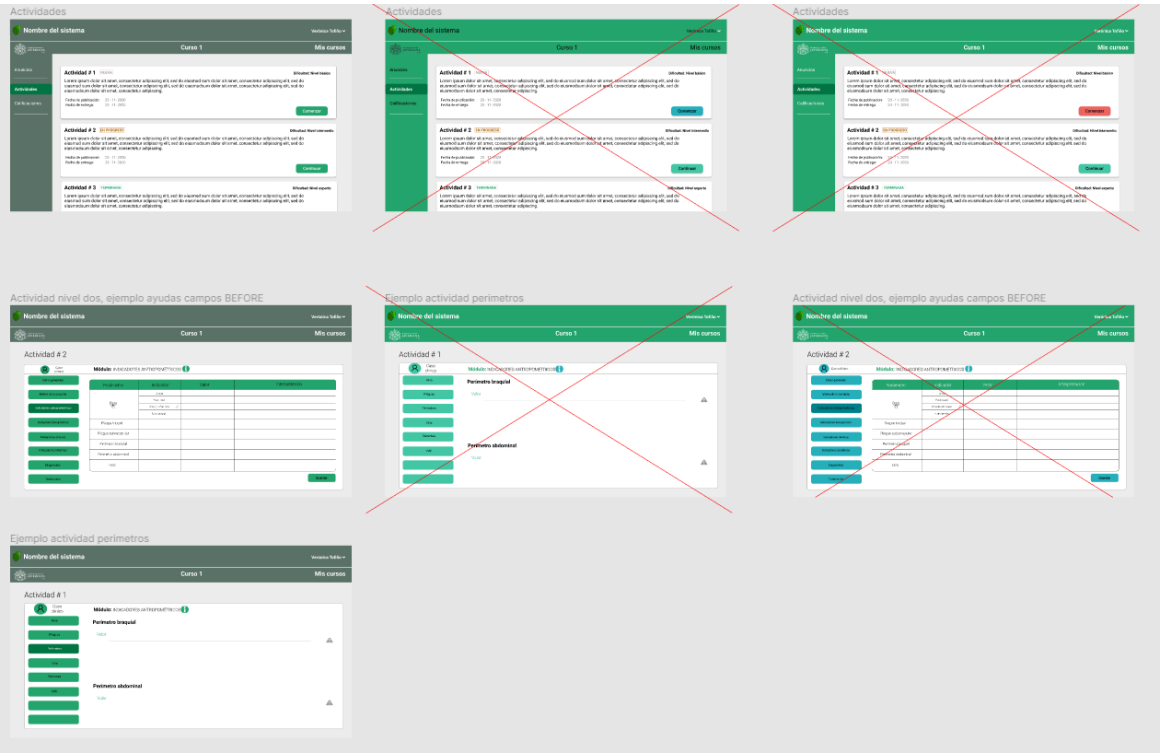



Figura 5.38: Diferentes propuestas de colores.

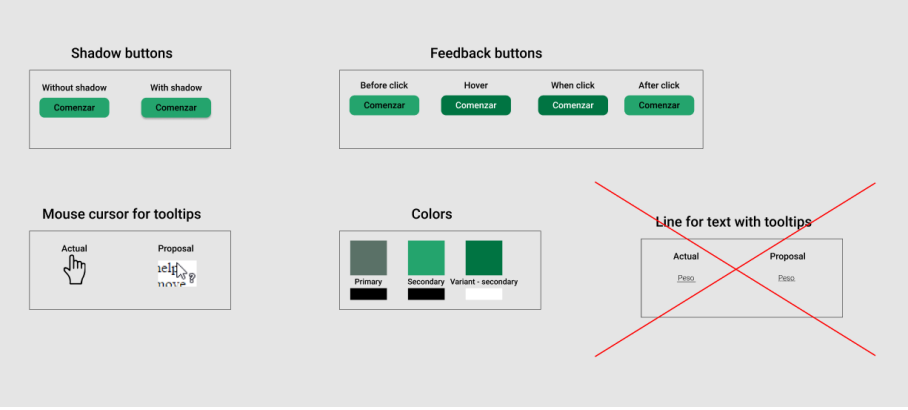



Figura 5.39: Propuesta final de mejoras.

Después de esto, se creó una subtarea que consistía en aplicar las mejoras escogidas a la aplicación real, la cual se llevó a cabo. Después de esa implementación, se dio por finalizada la tarea para mejorar la interfaz de usuario.

### 5.4.2. Notificaciones

Se decidió implementar notificaciones del sistema para que así el usuario tuviera retroalimentación inmediato sobre las acciones que realizaba. Para implementar estas notificaciones, se utilizó la librería Toastify, la cual permite mostrar diferentes tipos de notificación para el usuario.

A continuación en las figuras 5.40 y 5.41 se muestran ejemplos de que para ciertas acciones que realizaba el usuario, este recibía un mensaje de forma inmediata brindándole retroalimentación sobre si la acción quedaba registrada o había algún error en el proceso.

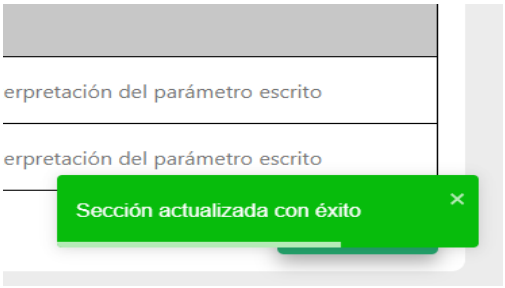



Figura 5.40: Ejemplo de notificación de éxito.

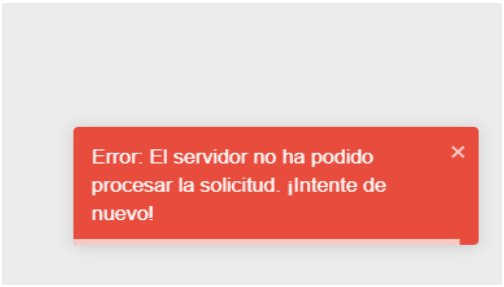



Figura 5.41: Ejemplo de notificación de fallido.

## 5.5. Mejoras de API

### 5.5.1. Ruta de los datos y las excepciones

La secuencia que siguen los datos que vienen de una petición es descrito por la figura 5.42:

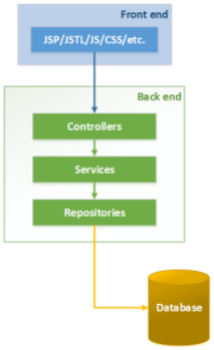



Figura 5.42: Secuencia de los datos a través de la aplicación.

Se entiende entonces que se reciben las peticiones provenientes del cliente, todo lo relacionado a la petición es manejado desde el controlador, *headers*, *queries* y *DTOs*. La siguiente etapa es pasar a los servicios donde se transforman los datos en objetos de negocio para aplicar lógica sobre ellos. Luego los datos pasarían a ser manipulados por los repositorios donde se manejan las colecciones de los datos para finalmente mandar a persistir los datos al servidor MongoDB.

Retomando lo que se mencionó previamente en la sección 5.3.2, se implementó un manejador personalizado de excepciones para centralizar el lugar donde estas se *catchean* para hacer el código más limpio y mantenible. Antes de que las peticiones HTTP lleguen al API ellas pasan por un filtro de *middlewares*, muchas veces los mismos frameworks con lo que se trabaja incluyen una serie de *middlewares bult-in*, es decir, incluidos en la plantilla del proyecto, que facilitan el desarrollo. La figura 5.43 expresa la idea de cómo las peticiones pueden pasar a través de varios *middlewares* antes de entrar y salir del API.

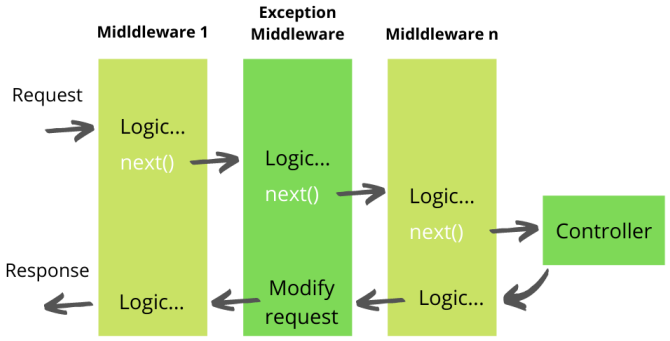
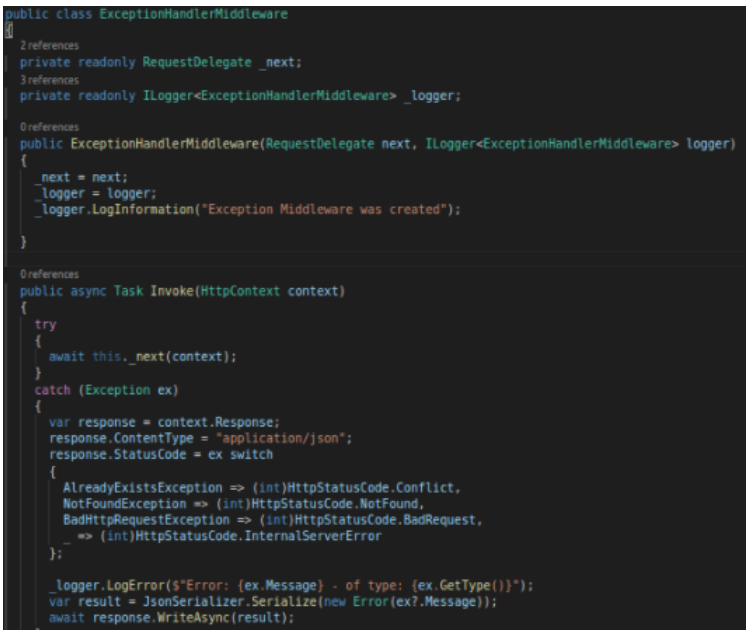



Figura 5.43: Ejemplo de filtros de *middlewares*.

En este proyecto se hizo uso de este concepto para evitar preocuparse al intentar atrapar todas las excepciones que se generen en el código y darle la respuesta más apropiada al cliente. Si en algún punto del código se realiza un *throw* la excepción va a ascender hasta alcanzar el *ExceptionHandlerMiddleware*, visto en la figura 5.44. Ahí es donde se le da un formato y un código HTTP al error que se le envía al cliente como respuesta según el tipo de excepción que se haya generado. Para especificar que se va a usar la clase como *middleware* basta con incluir una línea en el archivo *Startup.cs* para indicar que se quiere que la aplicación aplique a cada petición ese filtro.



```
public class ExceptionHandlerMiddleware
{
  2 references
  private readonly RequestDelegate _next;
  3 references
  private readonly ILogger<ExceptionHandlerMiddleware> _logger;

  0 references
  public ExceptionHandlerMiddleware(RequestDelegate next, ILogger<ExceptionHandlerMiddleware> logger)
  {
    _next = next;
    _logger = logger;
    _logger.LogInformation("Exception Middleware was created");

  }

  0 references
  public async Task Invoke(HttpContext context)
  {
    try
    {
      await this._next(context);
    }
    catch (Exception ex)
    {
      var response = context.Response;
      response.ContentType = "application/json";
      response.StatusCode = ex switch
      {
        AlreadyExistsException => (int)HttpStatusCode.Conflict,
        NotFoundException => (int)HttpStatusCode.NotFound,
        BadHttpRequestException => (int)HttpStatusCode.BadRequest,
        _ => (int)HttpStatusCode.InternalServerError
      };

      _logger.LogError($"Error: {ex.Message} - of type: {ex.GetType()}");
      var result = JsonSerializer.Serialize(new Error(ex?.Message));
      await response.WriteAsync(result);
```

Figura 5.44: *Middleware* manejador de excepciones.

Capítulo 6

# Pruebas

Dentro del ciclo de vida de un software, la fase de pruebas sirve para detectar errores que puede tener la aplicación antes de que está salga a producción, es decir, antes de que los usuarios finales encuentren el error y eso haga el sistema menos confiable. La fase de pruebas se realiza sobre todas las nuevas funcionalidades del sistema, antes de que estas se integren con la aplicación existente, para así poder corregir de forma temprana lo necesario, para así entregar un producto de calidad.

Actualmente existen muchos tipos diferentes de pruebas, los cuales prueban el comportamiento del sistema desde diferentes enfoques.

Para este proyecto de grado solo se tuvieron en cuenta tres tipos de pruebas, dos pruebas funcionales y una manual, las cuales serán explicadas a continuación.

## 6.1. Plan de pruebas

Para este proyecto de grado se siguió un plan de pruebas:

- Para probar la funcionalidad de la UI se realizaron pruebas unitarias, sobre toda la lógica y manejo de la información.
- Para probar la funcionalidad del API se realizaron pruebas unitarias y por componentes, sobre todos los controladores y sus respectivos servicios.
- Por último, se realizaron pruebas de usabilidad sobre usuarios reales, para probar, como ya se mencionó, la usabilidad de la aplicación.

### 6.1.1. Patrón AAA

Para desarrollar las pruebas unitarias y de componente del sistema se utilizó el patrón AAA, siglas para Arrange - Act - Assert en inglés. Este patrón sirve para estructurar una prueba y también para saber que se espera en cada paso de esta:

- *Arrange*: En este paso se prepara toda la información necesaria para poder realizar la ejecución de la prueba, por ejemplo realizar *Mocks* de servicios externos o cosas que se escapen del alcance de la prueba, al igual que información como datos.

- *Act*: En este paso se realiza la ejecución de la prueba, una vez se tiene la información lista.
- *Assert*: En este último paso se realizan las verificaciones sobre la prueba realizada para así certificar que la prueba se ejecutó de forma correcta y tiene el comportamiento esperado.

Con esa sencilla estructuración se pudo realizar las pruebas. Aunque todas las pruebas tienen implementando el patrón en algunas pruebas se ve de forma más explicita (como en la pruebas unitarias) y en otras no tanto (como las pruebas de componentes) pero implícitamente se sigue la misma lógica.

### 6.1.2. Herramientas

Tanto el front-end como el back-end se utilizaron diferentes librerías para la implementación de las pruebas.

#### 6.1.2.1. Jest

Las pruebas de la UI se realizaron utilizando Jest (librería Javascript para realizar pruebas), la cual tiene una gran integración con React, la tecnología del front-end utilizada.

La instalación fue relativamente sencilla, se instaló la librería por medio de npm y luego se siguieron los pasos de configuración requeridos para que el proyecto funcionara con Jest.

También se utilizó la nomenclatura destinada para las pruebas de Javascript, la cual dice que los archivos deben terminar en *.spec.js* o *.test.js* para que así la librería utilizada para las pruebas identifique cuales son los archivos de prueba para ejecutarlos.

#### 6.1.2.2. xUnit

Para el lado del API se utilizó xUnit (herramienta de prueba de unidad gratuita y de código abierto para .NET Framework) para realizar los tipos de pruebas definidos.

Siguiendo el patrón Clean, y cómo ya se explicó previamente, se creó un proyecto separado para las pruebas, llamado *Test*, en donde se instaló la dependencia requerida de xUnit.

## 6.2. Implementación de los tipos de pruebas

### 6.2.1. Pruebas unitarias

Las pruebas unitarias se realizan sobre las lógica que tienen un secciones de código, controladores, servicios, componentes (sea Back-end o Front-end). Este tipo de prueba sirven para probar funcionalidades específicas del sistema, en ambientes aislados, es decir sin la comunicación con servicios u otros componentes. Con esta capa se espera probar que la lógica del componente, de forma aislada, funciona de forma correcta y esperada. También ayuda a que, si se identifica algún error durante estas pruebas, se pueda resolver de forma rápida dado que al ser problema de la sección de

código que falla, se identifica inmediatamente que está mal. En este paso se utiliza el componente o servicio real instanciado de forma aislada (todas las funciones y llamado externos a el mismo *mockeadas*).

#### 6.2.1.1. UI

En la interfaz de usuario se realizaron pruebas unitarias sobre la secciones de código que manejan la lógica o comportamiento del sistema, los servicios y los *Slice redux* explicados en la sección anterior.

A continuación en la figura 6.1 se puede observar un ejemplo de prueba unitaria sobre el servicio de actividades.

```
1  import axios from "axios";
2  import { BASE_URL, PATH_ACTIVITES, PATH_FORMS } from "../../helpers";
3  import { getFormFromActivity, updateActivitySection, sendActivity } from "./activityService";
4
5  jest.mock("axios");
   Run | Debug
6  describe("activityService", () => {
7    const bodyNA = {···
120  };
121
122  const errorResponse = {···
125  };
126
127  afterEach(() => {
128    axios.get.mockClear();
129    axios.put.mockClear();
130    axios.post.mockClear();
131  });
132
   Run | Debug
133  it("should get the form an activity", async () => {
134    const sucessMessage = { success: true, data: bodyNA };
135    axios.get.mockImplementationOnce(() => Promise.resolve({ data: bodyNA }));
136
137    await expect(getFormFromActivity(1)).resolves.toEqual(sucessMessage);
138
139    expect(axios.get).toHaveBeenCalledWith(BASE_URL + PATH_ACTIVITES + 1 + "/form");
140  });
```

Figura 6.1: Ejemplo prueba unitaria servicio de actividades.

Aunque el patrón no está explícitamente definido, se puede identificar de forma sencilla. En la línea 3 se ve como se realiza el llamado a la función real del servicio de actividades, esto por lo explicado que se busca probar de forma aislada ese comportamiento. Por otro lado, en la línea 5 se ve como se realiza un *mock* de la librería axios (la utilizada para realizar peticiones HTTP) esto porque la librería no es lo que se busca probar y está fuera del servicio.

Volviendo a la prueba, el patrón se puede identificar de forma sencilla. En las líneas 134 y 135 se encuentra la preparación de la información de la prueba que se va a hacer (en este caso traer una actividad). Por lo que se prepara la respuesta exitosa y se realiza un *mockImplementation* sobre el llamado al método *GET*. Después, en la línea 137, se tiene el Act que es el llamado a la función del servicio de actividades para traer las actividades de un estudiante. Por último, en la línea 139 se realiza la verificación de que el método *GET* efectivamente fue llamado con los parámetros que se le pasaron en el act.

No se realizaron pruebas unitarias sobre los componentes dado que esos archivos contienen HTML. Además que no se utilizó el enfoque de Clases, sino de *Function* para implementar los componentes en React, por lo que no habían funciones de lógica individuales sino que se tocaba toca la función que incluía el HTML.

Realizando las pruebas unitarias sobre los servicios y *Slice redux* se pudieron encontrar pequeños errores que hubieran podido pasar desapercibidos. Por ejemplo, en la figura 6.2 se puede observar como al realizar la prueba unitaria sobre la visualización de errores del sistema, en el servicio de las actividades, no se estaba mostrando de una forma amigable para el usuario. Esto dado que un error 404, por ejemplo, no le dice mucho al usuario común, mientras que un mensaje en español con un poco más de información sí. (En este caso la prueba estaba configurada devolver un error cualquiera, se sabe que por convención el error 404 se refiere al término *Not found*).

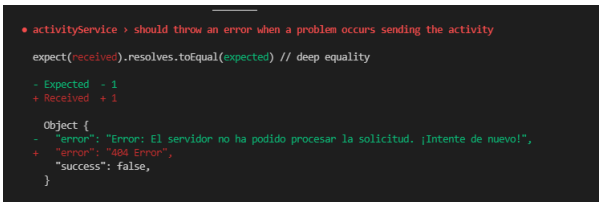



Figura 6.2: Retorno incorrecto del error.

Las pruebas no siempre deben indicar que algo está explícitamente mal, también pueden ayudar a dar más visibilidad sobre el comportamiento del sistema. En la figura 6.3 se puede observar como, por ejemplo, el llamado al *Endpoint* de traer las actividades de un estudiante había cambiado y al realizar la prueba no sé tenía conocimiento de esto.

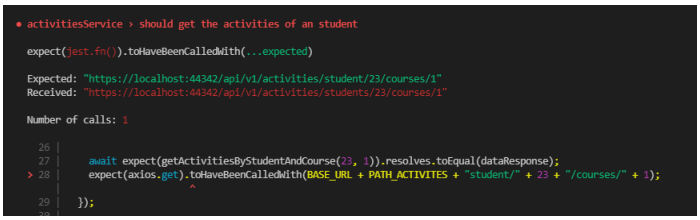



Figura 6.3: Llamado a *Endpoint* incorrecto.

Por último, en las figuras 6.4 y 6.5 se pueden observar otros pequeños cambios que las pruebas ayudaron a detectar. En la primera figura se ve como se estaba realizando mal el llamado del Redux, dado que la variable se llama *Form* y no *actions*, esto hacía que la variable *form* nunca se actualizara con el valor correspondiente. En la segunda imagen se ve como se estaba llamando mal al tipo de acción a realizar (se estaba diciendo que era de tipo *Setform*, pero se estaban actualizando las ayudas (*aids*).

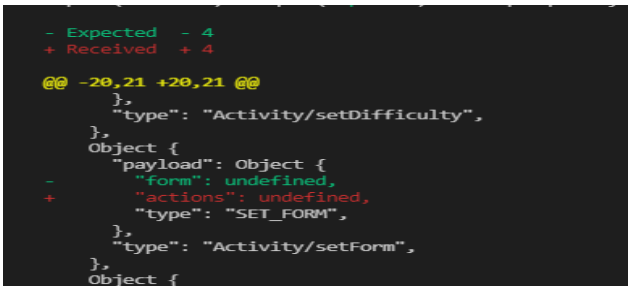



Figura 6.4: Mal definición de variable a actualizar.

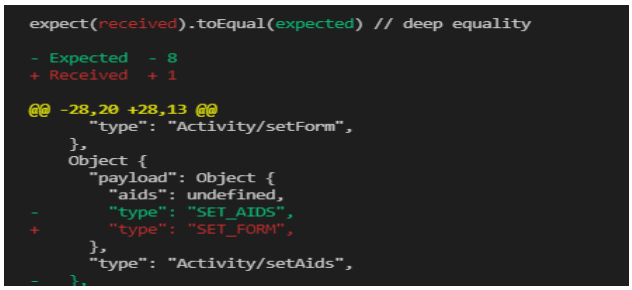
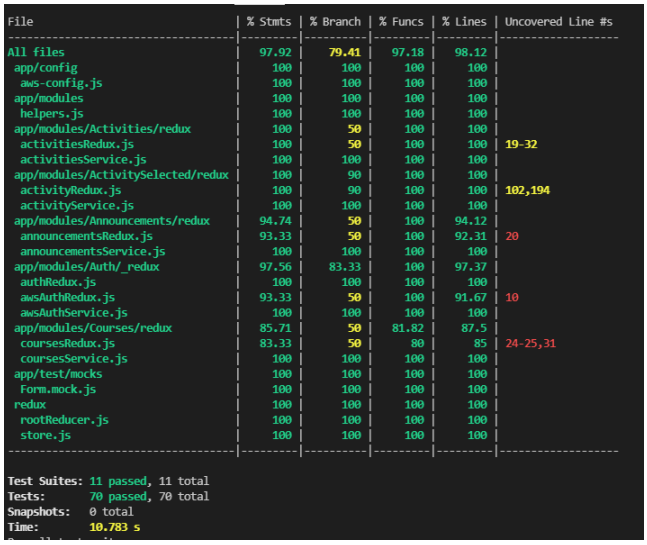



Figura 6.5: Mal llamado de tipo de acción.

Aunque a grandes rasgos, estos errores pueden ser muy sencillos, si se ponen en el contexto de que uno de estos errores pase mientras un usuario está utilizando la aplicación, el cual va a pasar por varias pantallas, diagnosticar que por ejemplo el fallo está en una mala definición de una variable, va a ser mucho más difícil de diagnosticar y corregir.

Como se dijo con anterioridad, las pruebas unitarias se realizaron sobre los servicios y slice redux. Como se ve en la figura 6.6 eso da un total de 70 pruebas unitarias, repartidas en todos los archivos de lógica de la aplicación. Otra ventaja que en general tienen las pruebas unitarias, y se pueden ver reflejado, es que corren relativamente rápido para la cantidad de pruebas, en este caso se demoran en promedio 10 segundos en correr las 70 pruebas.



| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
|---|---|---|---|---|---|
| All files | 97.92 | 79.41 | 97.18 | 98.12 | |
| app/config | 100 | 100 | 100 | 100 | |
| aws-config.js | 100 | 100 | 100 | 100 | |
| app/modules | 100 | 100 | 100 | 100 | |
| helpers.js | 100 | 100 | 100 | 100 | |
| app/modules/Activities/redux | 100 | 50 | 100 | 100 | |
| activitiesRedux.js | 100 | 50 | 100 | 100 | 19-32 |
| activitiesService.js | 100 | 100 | 100 | 100 | |
| app/modules/ActivitySelected/redux | 100 | 90 | 100 | 100 | |
| activityRedux.js | 100 | 90 | 100 | 100 | 102,194 |
| activityService.js | 100 | 100 | 100 | 100 | |
| app/modules/Announcements/redux | 94.74 | 50 | 100 | 94.12 | |
| announcementsRedux.js | 93.33 | 50 | 100 | 92.31 | 20 |
| announcementsService.js | 100 | 100 | 100 | 100 | |
| app/modules/Auth/_redux | 97.56 | 83.33 | 100 | 97.37 | |
| authRedux.js | 100 | 100 | 100 | 100 | |
| awsAuthRedux.js | 93.33 | 50 | 100 | 91.67 | 10 |
| awsAuthService.js | 100 | 100 | 100 | 100 | |
| app/modules/Courses/redux | 85.71 | 50 | 81.82 | 87.5 | |
| coursesRedux.js | 83.33 | 50 | 80 | 85 | 24-25,31 |
| coursesService.js | 100 | 100 | 100 | 100 | |
| app/test/mocks | 100 | 100 | 100 | 100 | |
| Form.mock.js | 100 | 100 | 100 | 100 | |
| redux | 100 | 100 | 100 | 100 | |
| rootReducer.js | 100 | 100 | 100 | 100 | |
| store.js | 100 | 100 | 100 | 100 | |

Test Suites: 11 passed, 11 total
Tests: 70 passed, 70 total
Snapshots: 0 total
Time: 10.783 s

Figura 6.6: Cubrimiento de los archivos de lógica de la UI.

#### 6.2.1.2. API

En el API se realizaron pruebas unitarias sobre los controladores y los servicios. Se decidió no hacer pruebas unitarias sobre los repositorios (dado que sería simular exactamente la función que provee el driver de MongoDB lo cual no es responsabilidad de este sistema) o sobre los controladores o servicios que llaman a los *mock servers* (dado que se espera que a futuro cambie para la integración definitiva con el otro sistema). En ese orden de ideas se realizaron las pruebas unitarias de todos los controladores que se enfocan en el desarrollo de una actividad y sus respectivos servicios.

Al igual que en la UI, las pruebas unitarias siguen el patrón AAA de forma implícita. A continuación en la figura 9.12 se muestra un ejemplo de una prueba unitaria realizada sobre el controlador de las actividades para al función de Crear una nueva actividad.

```
139     [Fact]
        0 references | Veronica Tofino, 14 hours ago | 1 author, 2 changes
140     public async Task CreateActivity()
141     {
142         const string activityId = "1";
143         var createActivityDto = new CreateActivityDto
144         {
145             Name = "Test Activity",
146             ActivityId = "activityId",
147             StudentId = "studentId",
148             Description = "Test description",
149             PublishDate = "02-02-2022",
150             Deadline = "10-02-2022",
151             DifficultyLevel = "2",
152             Sections = new List<string>()
153         };
154         var createdActivity = new Activity
155         {
156             Id = activityId,
157             Name = "Test Activity",
158             ActivityId = "activityId",
159             StudentId = "studentId",
160             Description = "Test description",
161             PublishDate = "02-02-2022",
162             Deadline = "10-02-2022",
163             DifficultyLevel = "2",
164             Sections = new List<string>()
165         };
166         _activitiesServiceMock.Setup(expression: a => a.CreateActivity(It.IsAny<Activity>())).ReturnsAsync(createdActivity);
167
168         var result = (ObjectResult) await _controller.CreateActivity(createActivityDto);
169
170         Assert.NotNull(result);
171         Assert.Equal(expected: (int) HttpStatusCode.Created, actual: result.StatusCode);
172         Assert.Equal(expected: result.Value, actual: createdActivity);
173         _activitiesServiceMock.Verify(expression: a => a.CreateActivity(It.IsAny<Activity>()), Times.Once);
174     }
```

Figura 6.7: Ejemplo de prueba unitaria para el controlador de actividades

En la anterior figura se puede ver como de la línea 142 a 166 se prepara la información necesaria, como la información de entrada y el *mock* al servicio de actividades (dado que se está probando el controlador) con la respuesta de ese llamado al servicio. En la línea 168 se realiza el llamado a la función real del controlador con la información de entrada y por último en las líneas 170 a 173 se realiza la verificación de que el controlador retorna el código esperado, la información esperada y que la función llamó al servicio correspondiente con la información correcta.

Realizando las pruebas unitarias sobre los controladores se pudo identificar que no todas las funciones estaban retornando el código de estado esperado. Este proyecto cuenta con cinco códigos de estado, acorde al estándar HTTP, para dar manejo a las respuestas del API. Estos códigos son: *Ok* (200), *Created* (201), *NotFound* (404), *Conflict* (409), *BadRequest* (400) y *InternalServerError* (500). Para el caso específico de creación de nuevas cosas se utiliza el código 201. Sin embargo, como se puede ver en la figura 6.8 tanto la creación de campos y sub-campos no estaban siguiendo el estándar definido:

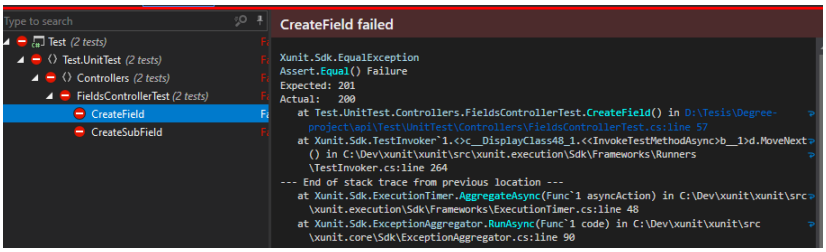



Figura 6.8: Error en el retorno de código de estado.

También, como el enfoque de las pruebas de unitarias es utilizar una instancia del componente o servicio real pero aislando su dependencia de otros archivos, se pudo ver que realizando la construcción de la instancia de algunos controladores se estaba realizando la construcción mal. Como se ve en la figura 6.9 aunque se estaba construyendo el controlador de *Form*, el *logger* se le pasaba una instancia del controlador de secciones, lo cual no es lo esperado. Realizando al construcción del controlador en la prueba se pudo identificar ese error.

```
public FormsController(
    ILogger<SectionsController> logger,
    IFormsService service)
{
    _logger = logger;
    _service = service;
    _logger.LogInformation(message: "Forms Controller was created!");
}
```

Figura 6.9: Construcción errónea del controlador.

Con estas pruebas se certifica que el funcionamiento de los archivos que manejan la lógica del sistema funcionan de forma esperada en ambientes aislados. En total se realizaron 30 pruebas unitarias como se puede observar en la figura 6.10, las cuales corren en aproximadamente 3 segundos.

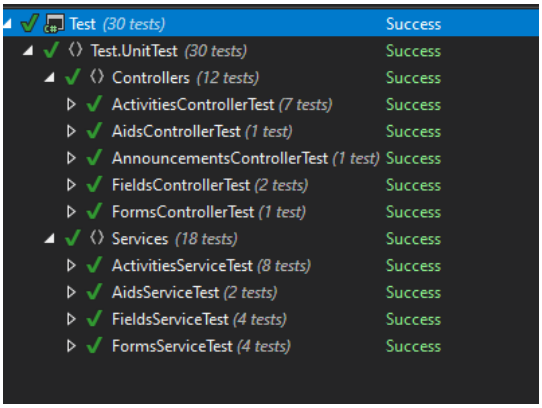



Figura 6.10: Cantidad de pruebas unitarias realizadas para el API.

### 6.2.2. Pruebas por componentes

Las pruebas por componentes se realizan sobre cada uno de los componentes de la aplicación de manera aislada. A diferencia de las pruebas unitarias, en este tipo de pruebas se realiza la integración de los controladores con su respectivos servicios y se baja hasta la capa de infraestructura, la cual para efectos de este tipo de pruebas, se utiliza un *Mock* o una Base de Datos en memoria dado que en las pruebas de componentes no se prueba la integración con otros sistemas o componentes, dado el caso. Este tipo de pruebas se realizan antes de las pruebas de integración, para verificar que, de forma aislada, cada parte del sistema construida en el proyecto tiene el comportamiento esperado. De igual forma estas pruebas son en sí la integración entre la comunicación de los controladores y los servicios, es por esto son menos y a su vez se busca verificar cosas que no se verificaron en las pruebas unitarias.

#### 6.2.2.1. API

Las pruebas de componentes en el back-end se diseñaron con la finalidad de probar las funcionalidades del núcleo en el que se enfoca el proyecto: la resolución de actividades. Por ende, las pruebas de componentes inician desde los controladores de actividades, historia clínica y campos de la historia clínica y terminan en los servicios que usan dichos controladores, se resalta que para los repositorios se utilizaron implementaciones falsas las cuales son posibles gracias a los paquetes de Moq y xUnit.

En el anexo #3 se muestra un poco como fue la construcción de estas pruebas, desde un enfoque técnico.

La carpeta destinada a las pruebas, al ser un proyecto separado, necesita incluir como paquetes los proyectos necesarios para que las pruebas funcionen y además, por como se implementaron las pruebas se necesita simular un cliente parecido que se crea en la clase de *Startup.cs*, por ello también se debe tener cuidado con el orden de la instanciación de *Singletons* para que ninguna dependencia sea necesitada antes de que su instancia sea creada. La primera vez que esto se probó aún no se había tenido en cuenta estos aspectos y se pudieron detectar mediante el resultado arrojado que exhibe la figura 6.11.


Figura 6.11: Descubrimiento de inyección de dependencias en el orden incorrecto e inclusión errónea de proyectos.

Hubo pruebas que ayudaron a encontrar errores en el código al proporcionar información de prueba sobre la que algunos métodos fallaban por factores como falta de validación para objetos nulos, falta de control sobre las excepciones y lógica innecesaria, más que todo en la limpieza de algunos bloques condicionales. Ejemplos de estos casos se pueden ver las figuras 6.12 y 6.13.

Con respecto a la primera figura particularmente (figura 6.12), esta prueba ayudó a analizar el caso de arrojar excepciones cuando el envío del correo al terminar la actividad falla, se consideró que al ser un servicio tercerizado y que es propenso a fallar en casos como rebote de correo por spam o situaciones de esa naturaleza, en vez de ser tratado con una simple excepción, debería de ser tratado como un evento que necesita reintentar ejecutarse una cierta cantidad de veces o necesita ser encolado para darle manejo después, también ayudó a hacer un análisis más individual en el tiempo límite que se le asignó a cada prueba para terminar de ejecutarse porque el envío de ese correo es una tarea que puede hacer que la acción de enviar una actividad sea más demorada de lo necesario y debería de tenerse en cuenta en su tiempo límite o *timeout*.

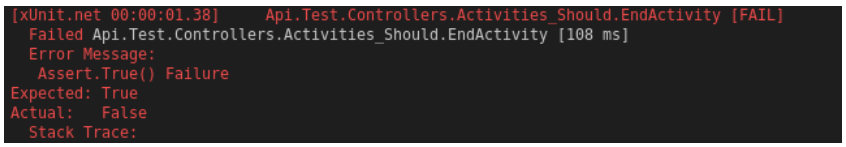



Figura 6.12: Prueba para la acción de enviar una actividad a revisión.

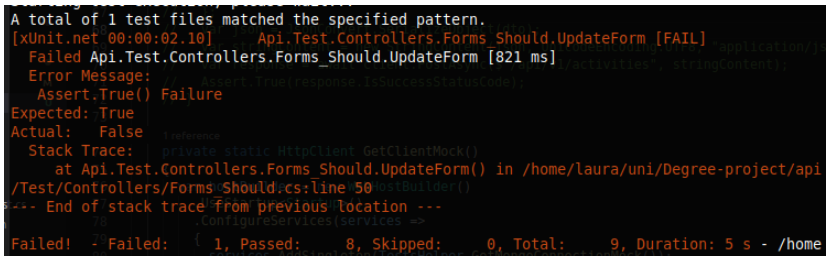



Figura 6.13: Prueba para la acción de guardar el progreso de una actividad.

Otro resultado interesante de las pruebas fue cuando se probó el *endpoint* encargado de traer actividades de un estudiante porque permitió percatarse de que en muchos lugares no se estaban haciendo conversiones seguras o *safe casts*. El no realizar este tipo de conversiones usando las palabras reservadas como *as* o *is*, se pueden generar excepciones inesperadas en el código que no tendrían que ver con la lógica de negocio sino con mal manejo de tipos. El resultado de la prueba puede verse en la figura 6.14.

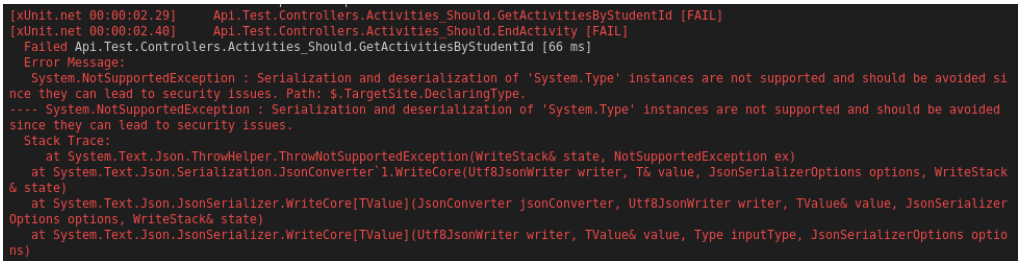



Figura 6.14: Prueba para la acción retornar las actividades de un estudiante.

Como se explicó con anterioridad, estas pruebas son menos y solo se realizaron sobre el flujo de actividades. Por lo que se tienen un total de ocho pruebas de componentes, como se muestra en la figura 6.15 las cuales corren en aproximadamente 6 segundos.

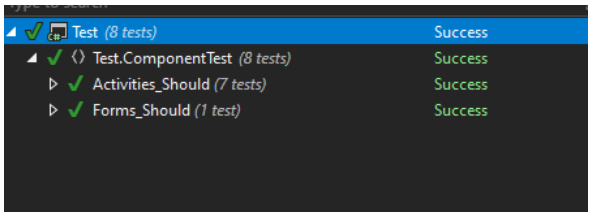



Figura 6.15: Total de pruebas de componente realizadas.

### 6.2.3. Pruebas de usabilidad

Las pruebas de usabilidad, son pruebas de tipo de manual, con las que se espera que los usuarios finales o afines prueben la interfaz de usuario, para que así puedan dar retroalimentación sobre formas en la que se podría mejorar. Con este tipo de pruebas se busca medir la experiencia de usuario por personas externas a los desarrolladores para así conocer cosas como: ¿Qué tan intuitiva es la aplicación? ¿Qué tan fácil de utilizar es? ¿Qué tan amigable es con el usuario? ¿Cómo cree que podría mejorar?. Una vez recolectada toda esta información, se puede analizar para así detectar posibles oportunidades de mejora del aplicación.

#### 6.2.3.1. Datos demográficos

Los usuarios que se eligieron para ser realizar estas pruebas cumplen con alguno de los dos tipos de perfil que se extrajeron de un análisis sobre los usuarios propensos a interactuar con la plataforma. El primer perfil es de un estudiante universitario con un nivel de conocimiento sobre los sistemas moderado que preferiblemente pertenezca a carreras como Nutrición y Dietética o Medicina, este es el perfil de la mayor parte del público objetivo en el que se centra la aplicación. El segundo perfil se refiere a un docente universitario con conocimiento moderado de los sistemas que preferiblemente se desenvuelva en disciplinas relacionadas a la salud. Se lograron realizar seis pruebas, la mitad de los entrevistados corresponden al primer perfil y la otra mitad al segundo perfil.

#### 6.2.3.2. UI

Para las pruebas de usabilidad en términos de de la interfaz de usuario se analizaron cuatro aspectos fundamentales en aproximadamente once tareas por usuario. Los aspectos que se evaluaron fueron: diseño fácil de entender, rapidez de desarrollo de tareas y navegabilidad, clicks por tarea y nivel de frustración o incomodidad. Los niveles de medición desde la primera categoría hasta la tercera son: Satisfactorio, Alto, Aceptable y Deficiente, para la última categoría, el nivel de frustración se mide en Bajo, Normal, Considerable y Alto. Los resultados, en general, fueron satisfactorios y más que todo para el público objetivo que son los estudiantes. Para los docentes se tuvo retroalimentación interesante para tener en cuenta. Los resultados pueden ser analizados en las siguientes figuras: 6.16, 6.17 y 6.18.

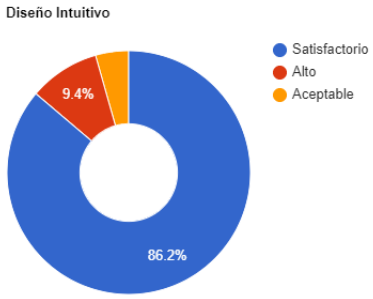



Figura 6.16: Resultados sobre la facilidad de los usuarios para entender el diseño en promedio.

Como se observa en la figura anterior, respecto a la facilidad de los usuarios para entender el sistema se tiene que un 86.2 % lo encuentran satisfactorio, un 4.4 % aceptable y un 9.4 % Alto, dando como resultado un rango favorable.

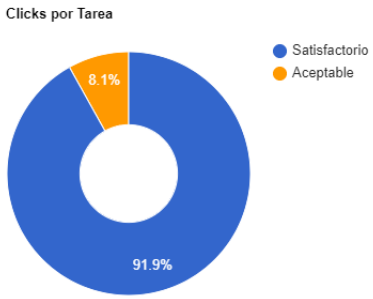



Figura 6.17: Resultados sobre las veces que los usuarios necesitan dar click para tareas recurrentes en promedio.

Como se observa en la figura anterior, respecto a la cantidad de clicks por tarea de los usuarios se tiene que un 91.9 % lo encuentran satisfactorio y un 8.1 % lo ve aceptable. Esto lo que expresa es que no se requieren muchas pantallas o clicks para cumplir con los objetivos del sistema, lo cual es positivo.

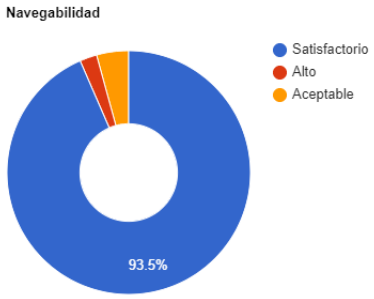



Figura 6.18: Resultados sobre la facilidad de navegabilidad para los usuarios en promedio.

Como se observa en la figura anterior respecto a la facilidad de navegabilidad del sistema se tiene que un 93.4 % lo encuentra satisfactorio y un 2.3 % alto lo que da a entender que los usuarios se pueden desplazar de forma sencilla y rápida por la aplicación.

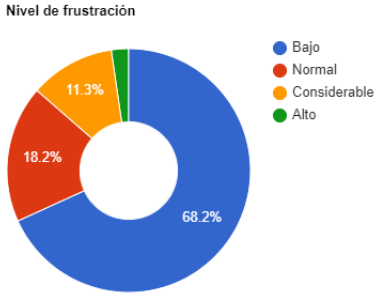



Figura 6.19: Resultados sobre el nivel de frustración o incomodidad de los usuarios en promedio.

Como se observa en la figura anterior respecto al nivel de frustración o incomodidad de los usuarios al usar la plataforma se tiene que el 68.2 % presenta un grado nulo de frustración, es decir, bajo. El 18.2 % presenta un nivel de frustración normal para ser la primera que interactúa con el sistema y recibe contexto sobre él. Luego en algunas tareas como la revisión de las ayudas el promedio del nivel de frustración de grado considerable subió a 11.3 % y en esa misma tarea se notó un pequeño porcentaje de usuarios con nivel de frustración alto, fue del 2.3 %, sin embargo, solo fue por un corto período de tiempo.

Gracias a estas pruebas se pudieron mejorar aspectos como: las leyendas de algunos *placeholders* para clarificar las interacciones posibles en la pantalla, reemplazar algunos iconos por otros más

intuitivos de asociar a su función real dentro de la plataforma, entender más qué esperan los usuarios de la plataforma, como por ejemplo funcionalidades de recuperar contraseña o mayor validación en los campos. Se mejoraron los mensajes de error y de éxito de acuerdo a los comentarios de los usuarios con respecto al tiempo de visibilidad de los errores en pantalla y de la semántica del mensaje. También los sujetos de prueba hicieron énfasis en que la interfaz se parece a la del sistema *Blackboard* que utiliza la universidad (en el cual se inspiró la interfaz de la plataforma), su curva de aprendizaje es bastante plana y, de hecho, tiene algunas características que les gustaría tener en el Blackboard, como por ejemplo el estado de una actividad que desarrolla un estudiante. Por último, se resalta que gracias a la retroalimentación de los usuarios se pudieron identificar aspectos para incluir en trabajo futuro para la plataforma que no se tenían previstos.

Capítulo 7

# Despliegue

El despliegue es una de las etapas finales del ciclo de desarrollo de software, seguida por la fase de monitoreo. La idea de esta fase del desarrollo es pasar el producto de software resultante de las fases de implementación y pruebas a un ambiente de producción. También es común desplegar la aplicación en ambientes de prueba o desarrollo dependiendo de qué tan rígida sea la cultura de pruebas dentro del equipo de desarrollo. Es una etapa crítica en la cual se debe tener extremo cuidado de no dañar datos, no realizar cambios de puertos, ni cualquier otro cambio sin medir antes las consecuencias y tener razones bien fundamentadas. Para estandarizar el proceso de despliegue y hacer más fácil el trabajo en equipo, se escribió un documento en el cual se describen los pasos para *dockerizar* cada servicio y desplegarlos exitosamente en el servidor (los pasos se pueden ver en el anexo #4). Se puede visitar el prototipo desplegado en la siguiente ruta: **http://18.116.106.255:3000/sign-in**.

## 7.1. Entorno

Como se mencionó anteriormente, se escogió la nube de AWS para realizar el despliegue del prototipo final. Para esto se creó una instancia de *Elastic Compute Cloud* (abreviado como EC2) en el AWS a la cual se le instaló Docker y Docker Compose para poder levantar los contenedores necesarios para la aplicación y acceder a ella desde la IP pública asignada a la instancia. A continuación se habla un poco sobre la configuración y proceso para realizar el despliegue del sistema.

### 7.1.1. EC2 y otros servicios de AWS

EC2 es un servicio de la nube de Amazon el cual permite la computación escalable dentro del AWS. Dentro de este servicio se crea la instancia de un servidor virtual el cual queda publicado dentro de la nube para poder ser accedido. Como se manejó la capa gratuita para este proyecto, con esta capa la IP pública de la instancia EC2 podía cambiar. Por eso, para lograr que la IP quedara fija, se utilizó una característica del EC2 llamada Direcciones IP Elásticas. Esta característica lo que hace es asignar una IP pública dentro de la piscina de IP's que tiene Amazon disponible, para que así se pueda tener una IP fija dentro del proyecto. Además de esto, fue necesario abrir los puertos dentro de la instancia EC2 que necesitan las aplicaciones, en este caso los puertos 27027 (BD), 5000 (API) y 3000 (UI), para poder realizar peticiones a los servicios del proyecto.

### 7.1.2. Docker y Docker Compose

Cómo ya se explicó en el capitulo anterior se utilizó Docker como herramienta para desplegar las imágenes de las aplicaciones (la base de datos MongoDB, el API y la UI). Para esto fue necesario instalar Docker dentro de la instancia EC2 creada. En el proyecto se utilizó docker-compose, *una herramienta para definir y ejecutar aplicaciones Docker multicontenedor. Con Compose, usted utiliza un archivo YAML para configurar los servicios de su aplicación. Luego, con un solo comando, se crea e inicia todos los servicios desde su configuración* [Doc]. Para esto, se crearon dos Dockerfile, uno para el API y otro para la UI. Estos archivos contienen una serie de pasos que sigue Docker para poder construir las imágenes requeridas y posteriormente subirlas al registro para bajarlas desde el EC2 y, eventualmente, levantar la aplicación.

## 7.2. Implementación

A continuación se explica de forma resumida como se realizó el despliegue de la base de datos, el API y la UI.

### 7.2.1. MongoDB

Lo primero que se decidió desplegar fue la base de datos, para esto se creó un archivo *docker-compose.yml* dentro del proyecto de API. Este archivo lo que hacía era bajar la imagen más reciente de mongoDB que Docker tuviera, luego de eso se configuraban los puertos donde se iba a correr la BD y el usuario y contraseña para poder ingresar a la base de datos. Por último, se subió el archivo docker-compose al servidor EC2 y se inició con el comando *docker-compose up*, una vez inicializado ya se pudo entrar a la base de datos de mongoDB creada dentro del servidor virtual.

### 7.2.2. API

El segundo servicio que se añadió en el despliegue fue el que corresponde al API. El Dockerfile que se diseñó sigue una estructura denominada *multistage*. La cuestión es ir construyendo la imagen por etapas y copiar solo lo necesario en cada etapa para no resultar con un *build* innecesariamente pesado.

Primero, se parte desde la imagen base del sdk de dotnet v.5: mcr.microsoft.com/dotnet/sdk:5.0, para dotar al contenedor de todo lo necesarios para correr la solución. Luego, se define un directorio y se procede a copiar los proyectos y demás archivos deseados de la solución en dicho directorio. Lo siguiente es publicar el proyecto de Api para generar su dll, puesto que será el punto de entrada a la aplicación. Finalmente, se construye la imagen con la configuración de *release* de la etapa anterior en la que se generó el dll.

### 7.2.3. UI

Lo último que se desplegó fue la interfaz de usuario. Para poder realizar este despliegue se configuró un *Dockerfile* dentro del proyecto de React, este archivo se llamó *Dockerfile.prod* por si a futuro se quiere desplegar otros tipos de ambientes se creen como *Dockerfile.ambiente*. Además de esto, se creó un archivo dentro de la solución llamado *.dockerignore*. Este archivo es para que Docker sepa que archivos no debe copiar cuando esté construyendo la imagen. En este *dockerignore* se agregaron todos los archivos referentes a pruebas, esto porque las pruebas no van a producción, son solo una herramienta para certificar el código que se está modificando. En este archivo también se agregó la carpeta de *node_modules* dado que, es más óptimo volver a correr y construir las dependencias de la aplicación que estar moviendo la carpeta de módulos de un sitio a otro.

En la figura 7.1 se muestra el *Dockerfile* y el *nginx.conf* para realizar el despliegue de la UI. Fue necesario dentro del archivo *Dockerfile* realizar dos pasos: el primer paso fue la construcción con las dependencias y archivos de la aplicación con npm (comando de node JS), después de esto se creó un *nginx server, un servidor web de código abierto que, desde su éxito inicial como servidor web, ahora también es usado como proxy inverso, cache de HTTP, y balanceador de carga* [kin21].

La instalación de Nginx se realizó para correr la aplicación dentro de este servicio. Como se ve en la figura 7.1 (linea 21 del Dockerfile), el archivo de construcción del app (llamado *builder*) se corre dentro de la carpeta html que tiene este servidor para poder levantar la aplicación en el ambiente de producción. También fue necesario dentro del proyecto agregar un archivo *nginx.conf* dado que era necesario realizar un redireccionamiento desde las rutas internas de la aplicación, para que estas pudieran ser utilizadas desde el servidor público nginx.

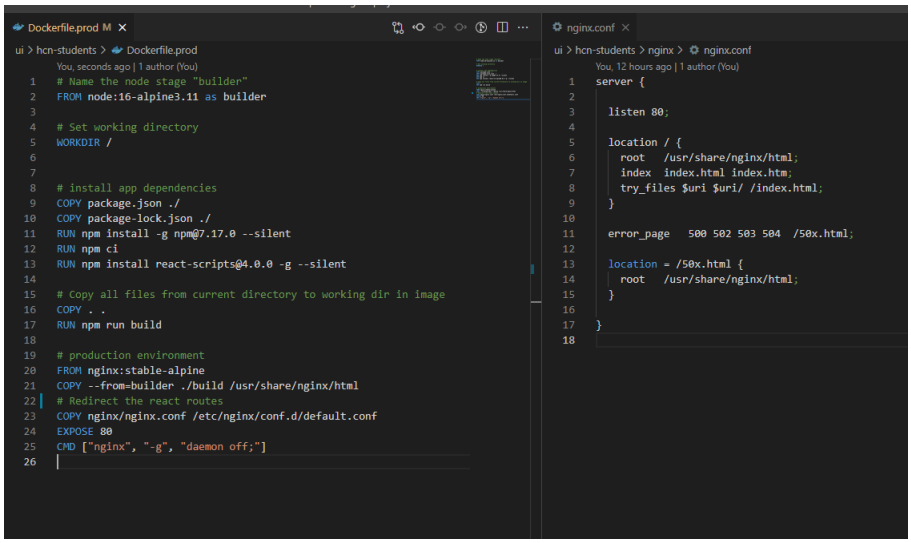



Figura 7.1: Configuración para desplegar la interfaz de usuario

Se utilizó la herramienta de Github de alojamiento de paquetes para subir la imágenes creadas por el Dockerfile con todo lo necesario para correr la UI y el API. Una vez publicados los paquetes, dentro del archivo docker-compose del EC2 se agregaron los servicios, con el nombre de la imagen con la que se publicó en Github y se reinició el Compose para que se levantaran los tres servicios, como se puede ver en la siguiente imagen.

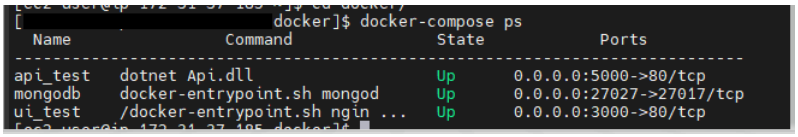



Figura 7.2: Despliegue de las tres aplicaciones

Capítulo 8

# Conclusiones y trabajo futuro

## 8.1. Conclusiones

Este trabajo tuvo como finalidad la construcción de un prototipo funcional de un simulador de historias clínicas nutricionales para los estudiantes de nutrición y dietética de la Pontificia Universidad Javeriana Cali. La construcción de este software se realizó siguiendo los pasos del ciclo de desarrollo de un software y con el uso de enfoques tecnológicos actuales los cuales pudieran suplir las necesidades del cliente.

El trabajo y prototipo presentando cumplen con lo esperado dado que cada decisión se tomó con fundamentos y previa investigación, para así poder dar cumplimiento a los requisitos definidos, dando como resultado un prototipo funcional el cual utiliza buenas prácticas y estándares de programación, además de tecnologías actuales, lo que hace que tenga soporte (tanto tecnológico como en ayudas por parte de comunidades).

Por medio de diversas herramientas, se logró que el sistema sea funcional de forma independiente, además de tener implementadas las bases para el trabajo futuro. Esto da como resultado un sistema con un buen grado de integrabilidad. Por otro lado, haber utilizado paradigmas y estándares de programación da como resultado un sistema con capacidad de ser extendido y mantenible a futuro.

El sistema está en la capacidad de que en el futuro cuando otras funcionalidades del sistema sean implementadas, como modificación de campos, eliminación o edición de secciones, entre otros, va a poder seguir trabajando como lo está haciendo actualmente sin necesidad de grandes cambios o refactorización de código. Esto es por el dinamismo que tiene la construcción de las historias clínicas nutricionales (tanto en el API como en la forma que la Interfaz de Usuario muestra esa HCN).

Un factor importante a resaltar es que este prototipo nació de la necesidad que presenta la carrera de nutrición y dietética para realizar el llenado de actividades de historias clínicas nutricionales por medio de un software. Sin embargo, la petición vino específicamente de una de las docentes de la carrera. En ese orden de ideas, una de las pruebas de usabilidad y a manera de "Demo" del prototipo se le realizó a la docente para conocer su opinión sobre si el sistema era el esperado para ella, al ser la cliente que lo había solicitado. La respuesta fue muy positiva dado que la cliente sintió que el software plasmaba una historia clínica real; cosas como el mostrar las secciones en el orden definido o campos con nombres reales y con información lo más real posible, dentro del desconocimiento

del equipo sobre temas de nutrición, la hicieron sentir muy apreciada como cliente. Por lo que se concluye que el sistema plasma lo esperado, que es simular historias clínicas reales, para su resolución por parte de los estudiantes de nutrición.

## 8.2. Trabajo futuro

Como todo sistema, este proyecto no queda implementado a la perfección y tiene varias cosas por ser mejoradas. El propósito de este capítulo es dejar algunas ideas que podrían ser útiles para continuar con su desarrollo del proyecto, recordando que este es un prototipo, y que provienen de la experiencia de haber trabajado en el proyecto, conocer el negocio y los alcances esperados de la plataforma.

### 8.2.1. Funcionalidad

Aunque actualmente se tiene un prototipo funcional, siempre hay oportunidad de mejora. Luego de realizar pruebas de la interfaz a algunos usuarios, se identificaron algunas oportunidades de mejora que se consideran ayudaran a brindar una mejor experiencia.

#### 8.2.1.1. Guardado automático y completo del progreso

Este es un punto de prioridad media a mejorar en el sistema prototipo dado que, como ya es de saber, el sistema permite guardar el progreso de las actividades sin necesidad de enviar la actividad para que sea calificada y poder retomar el progreso cuando el estudiante lo desee. Sin embargo, hay una forma particular de guardar el progreso y es por secciones, solo se guarda la sección que se está desarrollando actualmente y además, solo se guarda con el usuario presiona el botón con la leyenda de “Guardar”. Esta es una característica que de acuerdo a las pruebas de usabilidad realizadas, a los usuarios les resulta no esperada, se encuentra poco intuitiva y un poco molesta. La característica de guardado en sí, es de bastante ayuda para los estudiantes pero es recomendado mejorar la característica para cumplir con las expectativas y necesidades de los usuarios.

#### 8.2.1.2. Análisis de texto por medio de herramientas de Inteligencia artificial

Dado que este es un sistema para el aprendizaje, se necesita mejorar en la forma en la que el sistema brinda retroalimentación final a los usuarios. Es por esto que, por medio de inteligencia artificial, se podría realizar un análisis de texto sobre las respuestas del usuario para así comparar acorde a lo esperado. Esto ayudaría a brindar una retroalimentación más personalizada sobre lo que escriben los estudiantes y lo que se esperaba que debían hacer.

### 8.2.2. Back-end

Dentro de los trabajos futuros para el back-end se escriben los que se consideran deberían implementarse para así completar o mejorar el flujo de la información que maneja el sistema

#### 8.2.2.1. Integración con la plataforma de docentes

La versión entregada del prototipo funciona consumiendo la información que extraería de la plataforma de docentes de un servidor mock, es decir, no hay una integración definitiva entre ambos sistemas. Por el momento ambas plataformas están completamente separadas y la idea es que a futuro se integren siguiendo la arquitectura aquí planteada, la cual soporta el paso de información entre ambas plataformas. Entre las alternativas que se estudiaron se encuentra el uso de las peticiones dentro del esquema REST o un sistema de colas para duplicar la información de la plataforma de docentes en las bases de datos en la plataforma de estudiantes, siguiendo las estrategias más comunes para el manejo de servicios, otra alternativa es centralizar las bases de datos de ambas plataformas en un único set. Darle resolución a este punto es de máxima prioridad para completar un producto de valor mínimo que se pueda poner en funcionamiento para la universidad, pero se resalta que las opciones aquí planteadas son solo recomendaciones, más no es un camino obligatorio. Pueden haber estrategias de integración diferentes que valen la pena ser analizadas.

#### 8.2.2.2. Procedimientos con timers

El prototipo actualmente muestra solo las actividades que tienen una fecha límite o menor a hoy. Sin embargo, las actividades que el estudiante nunca envió al docente y ya están pasadas de la fecha actual no son enviadas por el sistema o se actualiza su estado. Esto hace que muchas actividades queden de forma indefinida en estado Nueva o En progreso, una mejora interesante podría ser tener procedimientos que se realicen cada cierto tiempo que ayuden moviendo esas actividades que el usuario no envió manualmente, ya sea borrándolas del sistema para que no genere ruido o enviándolas para ser calificadas (se debe evaluar cual debería ser el comportamiento esperado).

### 8.2.3. Front-end

Dentro de los trabajos futuros para el front-end se han compilado una serie de propuestas para mejorar algunos aspectos para reforzar la mantenibilidad y extensibilidad del código.

#### 8.2.3.1. Preprocesadores CSS

En la actualidad el uso de preprocesadores CSS es considerado una buena práctica para desarrollo front-end. Primero, porque permite extender las características del CSS puro e implementar lógica que contribuye a la limpieza y reusabilidad del código como mixins y herencia. Y, segundo, permite organizar mucho mejor el código CSS en funciones y variables haciendo posible, por ejemplo, cambiar la paleta de colores de la aplicación tan solo cambiando el valor de las variables de color necesarias. Razones por las cuales es una opción para considerar a futuro.

#### 8.2.3.2. Herramienta de construcción de paquetes

El levantamiento de la aplicación, por parte de la UI, es relativamente demorado dado que se deben construir e incluir en la aplicación, todas las dependencias (node-modules) que utiliza el

sistema para funcionar. Por ejemplo: Redux, Bootstrap, Material Icon, Toastify, entre otros.

Al ser muchas dependencias dentro del sistema el levantamiento de la aplicación es demorado. Para mejorar este rendimiento, se puede incluir a futuro una herramienta de construcción de paquetes. Esto ayudará a que se haga una construcción previa sobre las dependencias y paquetes que necesita el sistema para funcionar, y una vez siga al paso de correr ese paquete construido, el levantamiento se haga en cuestión de segundos.

### 8.2.4. DevOps

Los trabajos que son considerados pertenecientes a la disciplina de Development & Operations están más relacionados al empaquetamiento, publicación y despliegue de la aplicación, también se incluyen algunas mejoras que pueden ser vistas como deuda técnica.

#### 8.2.4.1. Pipeline

Seguir un proceso estandarizado para integrar cambios, versionar el API o la SPA y finalmente desplegar nuevas versiones de la aplicación es vital para los sistemas propensos a crecer. Particularmente el sistema implementado en este proyecto tiene una gran proyección y se considera que una tubería para automatizar pequeños procesos como por ejemplo, ejecutar los análisis estáticos y dinámicos para verificar que el código cumple con los estándares impuestos por la librearía ESLint y para verificar el cumplimiento de determinado grado de cobertura en cuanto a las pruebas funcionales. En adición a lo anterior, mediante una o varias tuberías también se podrían delegar responsabilidades como optimizar el código antes de empaquetarlo y eventualmente desplegar la aplicación, esto por supuesto acompañado del uso de algún sistema de manejo de versiones para controlar cuándo desplegar el código y elegir qué código será desplegado.

### 8.2.5. Pruebas

Dentro del sistema se realizaron pruebas de unidad para el API y la UI y pruebas de componente para el API. A continuación se expresan otro tipos de pruebas que se consideran podrían brindar valor al sistema.

#### 8.2.5.1. Pruebas de componente UI

Por temas de tiempo no fue posible implementar las pruebas de componente de la UI, sin embargo se conoce su valor al tener la responsabilidad de probar que los componentes que ve el usuario se comporten como debe de ser en por ejemplo, la forma en que se visualizan campos, las acciones que resultan de los eventos realizados por usuarios (como clicks), entre otros. Es por esto que se considera importante realizar estas pruebas sobre las páginas de la aplicaciones y así certificar que el sistema se comporta de forma esperado para cada una de las partes que contiene.

#### 8.2.5.2. Pruebas de integración

Las pruebas de componentes se pueden ver como una clase de prueba de integración, dado que prueba el comportamiento de los componentes y servicios en conjunto para ver que la comunicación entre ambos funciona bien. Sin embargo, esas pruebas no cubren como se comporta el sistema desarrollado con sistemas externos como bases de datos, colas de mensajes y otras aplicaciones. Dado que, por ejemplo el API y la UI se realizan en proyectos a parte, sería bueno definir la estrategia de pruebas de integración para verificar la comunicación entre ambos sistemas. También la comunicación del API con el módulo de docentes o la base de datos u otro sistema externo que se considere brindaría valor.

# Bibliografía


[3D 15] 3D PROTOTYPE DESIGNER. *What Is A Functional Prototype*. Abr. de 2015. URL: https://3d-printing-expert.com/what-is-a-functional-prototype/ (visitado 23-04-2020).

[Aca] Academy of Nutrition and Dietetics. *The Nutrition Care Process (NCP)*. URL: https://www.ncpro.org/nutrition-care-process (visitado 16-10-2020).

[ACM12] ACM. *Computing Classification System*. 2012. URL: https://dl.acm.org/ccs (visitado 24-04-2020).

[AWS] AWS. *AWS | Gestión de identidades y autenticación de usuario en la nube*. URL: https://aws.amazon.com/es/cognito/.

[Bar] Abimael Barea. *Clean Architecture | Deloitte España | Tecnología*. URL: https://www2.deloitte.com/es/es/pages/technology/articles/clean-architecture.html.

[C4M] C4Model. *The C4 model for visualising software architecture*. URL: https://c4model.com/ (visitado 20-12-2020).

[Cal] Calameo. *Metodologia De Roger Pressman*. URL: https://es.calameo.com/read/00570389414c8ee17fd43 (visitado 20-12-2020).

[CC04] José Luis Castillo Hernández y Roberto Zenteno Cuevas. "Valoración del Estado Nutricional". En: *Revista Médica de la Universidad Veracruzana* 4.2 (dic. de 2004). URL: https://www.medigraphic.com/pdfs/veracruzana/muv-2004/muv042e.pdf.

[Cen] Center for Healthy Living. *Simulation Lab*. URL: https://sph.uth.edu/research/centers/dell/nourish-program/simulation-lab.htm (visitado 03-05-2020).

[DEV20] DEVELOPMENT APPLICATION JAVA. *What Is a Modular Monolith? | Rebel*. Jun. de 2020. URL: https://www.jrebel.com/blog/what-is-a-modular-monolith (visitado 20-12-2020).

[Doc] Docker. *Overview of Docker Compose | Docker Documentation*. URL: https://docs.docker.com/compose/.

[Eca16] Yván Ecarri. *Estilos y Patrones de Arquitectura de Software – oSoft Blog*. Mar. de 2016. URL: http://blog.osoft.es/index.php/2016/03/24/estilos-y-patrones-de-arquitecturas-de-software/.

[Eel06] Peter Eeles. *What is a software architecture?* Feb. de 2006. URL: https://www.ibm.com/developerworks/rational/library/feb06/eeles/index.html (visitado 23-04-2020).

[Flo15] Martin Flower. *MonolithFirst*. Jun. de 2015. URL: https://martinfowler.com/bliki/MonolithFirst.html.

[Gue16] Manuel Guerrero. *Agile Lean Manufacturing: ¿qué son los Story Points? - Kaizen, Mejora Continua*. Ago. de 2016. URL: https://manuelguerrerocano.com/agileleanmanufacturing_storypoints/.

[Gur+10] M Gurinović y col. *EURRECA nutritional planning and dietary assessment software tool: NutPlan. - PubMed - NCBI*. 2010. URL: https://www.ncbi.nlm.nih.gov/pubmed/20517319 (visitado 03-05-2020).

[IBM] IBM. *What is software development? | IBM*. URL: https://www.ibm.com/topics/software-development.

[ING18] INGSOFTWARE. *Software Prototyping - Ingsoftware*. Jun. de 2018. URL: https://www.ingsoftware.com/software-prototyping (visitado 23-04-2020).

[Jor16] Laura Jorge. *El estudio antropométrico, ¿para qué sirve?* Abr. de 2016. URL: https://laurajorgenutricion.com/el-estudio-antropometrico-para-que-sirve/ (visitado 16-10-2020).

[kin21] kinsta. *¿Qué Es NGINX y Cómo Funciona? NGINX explicado para principiantes*. Abr. de 2021. URL: https://kinsta.com/es/base-de-conocimiento/que-es-nginx/.

[Lat10] Fatimah Lateef. "Simulation-based learning: Just like the real thing". En: *Journal of Emergencies, Trauma and Shock* 3.4 (oct. de 2010), págs. 348-352. ISSN: 09742700. DOI: 10.4103/0974-2700.70743. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2966567/.

[Lea] LearningPool. *The five levels of software simulation for e-learning - Learning Pool*. URL: https://learningpool.com/the-five-levels-of-software-simulation-for-e-learning/ (visitado 16-10-2020).

[Mic19] Microsoft. *Estilo de arquitectura de microservicios - Azure Application Architecture Guide | Microsoft Docs*. Oct. de 2019. URL: https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices (visitado 23-04-2020).

[Min04] MinEducación. *Una llave maestra Las TIC en el aula - ..::Ministerio de Educación Nacional de Colombia::..* 2004. URL: https://www.mineducacion.gov.co/1621/article-87408.html (visitado 03-05-2020).

[Nie12] Jakob Nielsen. *Usability 101: Introduction to Usability*. Ene. de 2012. URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (visitado 23-04-2020).

[Nut19] Nutrium. *Nutrition analysis software for students and researchers*. 2019. URL: https://blog.nutrium.io/students-researchers-nutrium-50-off/ (visitado 03-05-2020).

[Pon16] Pontificia Universidad Javeriana Cali. *Nutrición y Dietética | Pontificia Universidad Javeriana, Cali*. 2016. URL: https://www.javerianacali.edu.co/programas/carreras/nutricion-y-dietetica (visitado 03-05-2020).

[Riv+18] Etienne Rivière y col. "Twelve tips for efficient procedural simulation". En: *Medical Teacher* 40.7 (jul. de 2018), págs. 743-751. ISSN: 1466187X. DOI: 10.1080/0142159X.2017.1391375. URL: https://www.tandfonline.com/doi/abs/10.1080/0142159X.2017.1391375.

[SA95] Ramón S. Salas Perea y Plácido Ardanza Zulueta. *La simulación como método de enseñanza y aprendizaje*. 1995. DOI: ISSN0864-2141. URL: http://scielo.sld.cu/scielo.php?script=sci%7B%5C_%7Darttext%7B%5C&%7Dpid=S0864-21411995000100002 (visitado 03-05-2020).

[Sán16] Jorge Sánchez. *¿Por qué utilizo clean Architecture?* Jul. de 2016. URL: https://xurxodev.com/por-que-utilizo-clean-architecture-en-mis-proyectos/.

[SH10] A Suverza y K Haua. *El Abcd De La Evaluación Del Estado De Nutrición*. Ed. por Javier de León. Primera ed. Vol. 67. 6. Santa fe: Mc Graw Hill, 2010, págs. 14-21. ISBN: 9786071503374. URL: https://issuu.com/jcmamanisalinas/docs/el%7B%5C_%7Dabcd%7B%5C_%7Dde%7B%5C_%7Dla%7B%5C_%7Devaluaci%7B%5C_%7D%7B%5C_%7Dn%7B%5C_%7Ddel%7B%5C_%7Destad.

[SSP04] Araceli Surveza, Adriana Salinas y Otilia Perichart. *HISTORIA CLÍNICO-NUTRIOLÓGICA*. Inf. téc. Ciudad de México: Universidad Iberoamericana, ene. de 2004. URL: https://ibero.mx/campus/publicaciones/clinica%7B%5C_%7Dnutric/pdf/Documentonormativo.pdf.

[Tha] Dinesh Thakur. *What is Software Requirement? Types of Requirements. - Computer Notes*. URL: https://ecomputernotes.com/software-engineering/softwarerequirement (visitado 07-11-2020).

[The17] The Segue Creative Team. *Why is Usabilty Important to Application Development?* Dic. de 2017. URL: https://www.seguetech.com/usability-application-development/ (visitado 23-04-2020).

[Tro+18] Claudia Troncoso Pantoja y col. "Design of an electronic clinical record simulator for Nutrition and Dietary students". En: *Educacion Medica* 19 (nov. de 2018), págs. 238-245. ISSN: 15751813. DOI: 10.1016/j.edumed.2017.09.006. URL: https://reader.elsevier.com/reader/sd/pii/S1575181317301742?token=https://www.sciencedirect.com/science/article/pii/S1575181317301742.

[Wik20a] Wikipedia. *Instructional simulation*. Mar. de 2020. URL: https://en.wikipedia.org/wiki/Instructional%7B%5C_%7Dsimulation (visitado 16-10-2020).

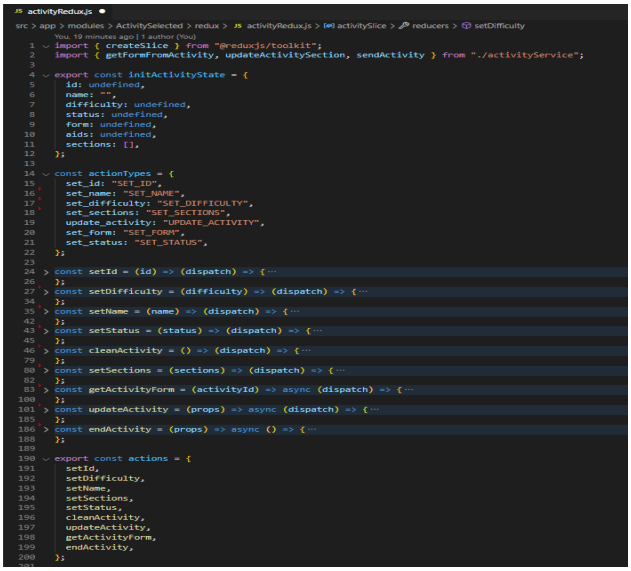
[Wik20b] Wikipedia. *Software prototyping*. Feb. de 2020. URL: https://en.wikipedia.org/wiki/Software%7B%5C_%7Dprototyping%7B%5C#%7Dcite%7B%5C_%7Dnote-1 (visitado 23-04-2020).

Capítulo 9

# Anexos

## Anexo #1, Implementación de Redux

A continuación se explica un poco el proceso de guardado y acceso de información con Redux:



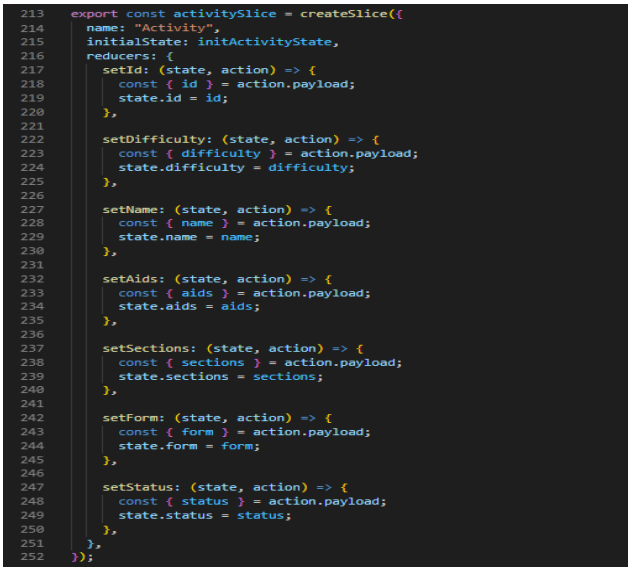




Figura 9.1: Primera parte creación slice

Figura 9.2: Segunda parte creación slice

Como se muestra en las figuras 9.1 y 9.2 primero se debe crear el *slice* con la información que se desea guardar. También se definen las funciones que puede realizar dicho *slice* para modificar la información. Un *slice* normalmente consta de 4 partes:

- En primera instancia se declara la variable *initState*, la cual define cuáles son las variables que va a guardar el *slice*.
- Luego se definen los *actionTypes*, los cuales son los tipos de acción que puede ejecutar el slice.
- En la parte de *actions* se definen las funciones del *slice*. Cada una de las funciones debe llamarse igual tanto en la variable de *actions*, como donde se implementa, porque sino al llamar a la función por medio de *actions*, este no podrá encontrarla y ejecutarla.

- Por último se define el *slice* (figura 9.2) el cual se encarga en sí de las actualizaciones de estado de las variables, este *slice* también es el que se definirá después en la *store* que es donde se almacenan todos los *slices* para que así puedan ser llamados desde cualquier parte del código.

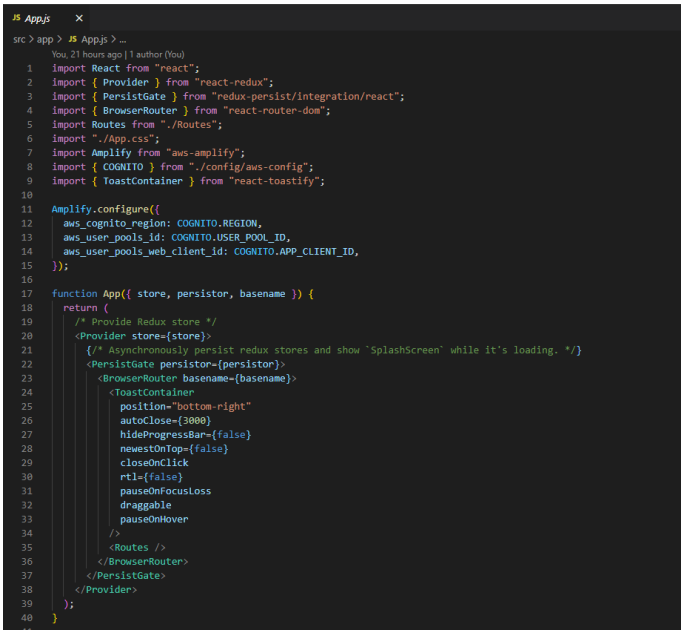
Una vez se tenga la creación del o los *slices* necesarios, se deben configurar en la *store*, como se dijo antes, para que así puedan ser accedidos en todas las secciones del código.

```
import { combineReducers } from "redux";
import { activitySlice } from "../app/modules/ActivitySelected/redux/activityRedux";
import { activitiesSlice } from "../app/modules/Activities/redux/activitiesRedux";
import { announcementSlice } from "../app/modules/Announcements/redux/announcementsRedux";
import { sidebarSlice } from "../app/modules/Sidebar/redux/sidebarRedux";

import * as auth from "../app/modules/Auth/_redux/authRedux";

export const rootReducer = combineReducers({
  auth: auth.reducer,
  activities: activitiesSlice.reducer,
  sidebar: sidebarSlice.reducer,
  activity: activitySlice.reducer,
  announcement: announcementSlice.reducer,
});
```

Figura 9.3: Uso de los componente Modal

En la figura 9.3 se muestra la definición de la *store* con todos los *slices* que cuenta la aplicación, incluido el de *activity* el cual se explicó en el paso anterior.

Una vez creada se debe agregar la *store* como dependencia de todos los componentes, como se muestra a continuación.



```
App.js
src > app > App.js > ...
You, 21 hours ago | 1 author (You)
import React from "react";
import { Provider } from "react-redux";
import { PersistGate } from "redux-persist/integration/react";
import { BrowserRouter } from "react-router-dom";
import Routes from "./Routes";
import "./App.css";
import Amplify from "aws-amplify";
import { COGNITO } from "./config/aws-config";
import { ToastContainer } from "react-toastify";

Amplify.configure({
  aws_cognito_region: COGNITO.REGION,
  aws_user_pools_id: COGNITO.USER_POOL_ID,
  aws_user_pools_web_client_id: COGNITO.APP_CLIENT_ID,
});

function App({ store, persistor, basename }) {
  return (
    /* Provide Redux store */
    <Provider store={store}>
      {/* Asynchronously persist redux stores and show `SplashScreen` while it's loading. */}
      <PersistGate persistor={persistor}>
        <BrowserRouter basename={basename}>
          <ToastContainer
            position="bottom-right"
            autoClose={3000}
            hideProgressBar={false}
            newestOnTop={false}
            closeOnClick
            rtl={false}
            pauseOnFocusLoss
            draggable
            pauseOnHover
          />
          <Routes />
        </BrowserRouter>
      </PersistGate>
    </Provider>
  );
}
```

Figura 9.4: Uso de los componente Modal

Como se muestra en la figura 9.4 la configuración de este proyecto tiene como componente padre

el componente App, por lo cual antes de este se define que el padre es el *Provider* con la *store*, para así inyectar dentro de todos los componentes el llamado y actualización de los estados definidos en los *slices*.

Una vez configurada e inyectada la *store*, se puede realizar la actualización de estados o variables configurados en los *slices*, en cualquier parte del código.

```
73          await dispatch(actionsActivitySelected.getActivityForm(card.id)).then((response) => {
74            if (mounted) {
75              if (response.success) {
76                setShowModalInfoW(false);
77                dispatch(actionsSidebar.setVisibility(false));
78                dispatch(actionsActivitySelected.setId(card.id));
79                dispatch(actionsActivitySelected.setSections(card.sections));
80                dispatch(actionsActivitySelected.setName(card.name));
```

Figura 9.5: Uso de dispatch

```
18  const { sections, name, status } = useSelector((state) => state.activity);
19
20  const closeActivity = (e) => {
21    e.preventDefault();
22    status === "Calificada" ? history.push("/feedbacks") : history.push("/activites");
```

Figura 9.6: Uso de selector

Como se muestra en las figuras 9.5 y 9.6 la actualización y uso de las variables se realizan por medio de *dispatch* y *selector*, funciones propias de Redux, respectivamente. *Dispatch* sirve para actualizar o cambiar los estados de los *slices* con nueva información. Por otro lado, con selector se puede traer el estado actual de las variables del Slice para poder realizar acciones con ellas, como se muestra en la figura 9.6.

## Anexo #2, Implementación de AWS Cognito

A continuación, en las figuras 9.7, 9.8 y 9.9 se explica de forma resumida la conexión con React para el manejo de usuarios por medio de Cognito:

```
JS aws-config.js ×
src > app > config > JS aws-config.js > ...
        You, 6 days ago | 1 author (You)
1   export const COGNITO = {
2     REGION: [redacted]
3     USER_POOL_ID: "[redacted]",
4     APP_CLIENT_ID: "[redacted]",
5   };
```

Figura 9.7: Configuración credenciales Cognito.

Luego de la instalación del plugin, se creó un archivo llamado *aws-config* en donde se guardaron las credenciales dadas por AWS Cognito para generar la conexión al repositorio de usuarios.

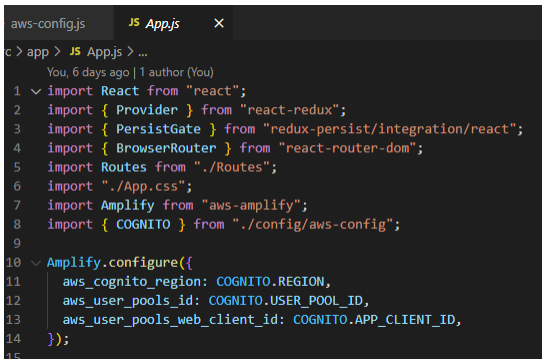



Figura 9.8: Conexión con AWS Amplify.

Después dichas credenciales se agregan al AWS Amplify para generar la conexión:

```
import { Auth } from "aws-amplify";

export const signInAWS = async (email, password) => {
  try {
    const user = await Auth.signIn(email, password);
    return { success: true, user: user };
  } catch (err) {
    return { success: false, error: err.message };
  }
};

export const signUpAWS = async (email, password, attributes) => {
  try {
    await Auth.signUp({
      username: email,
      password: password,
      attributes: attributes,
    });
    return { success: true, error: null };
  } catch (err) {
    return { success: false, error: err.message };
  }
};
export const confirmAccountAWS = async (email, password) => {
  try {
    await Auth.confirmSignUp(email, password);
    return { success: true, error: null };
  } catch (err) {
    return { success: false, error: err.message };
  }
};
```

Figura 9.9: Llamado a los servicios de AWS.

Por último, se utiliza la función *Auth* de la librería para el llamado de los *endpoints* de inicio de sesión, creación de cuenta y confirmación de cuenta.

Con esta integración se tiene un manejo completo de usuarios. Los *endpoints* devuelven mensajes de error si las credenciales no son correctas, o el correo ya existe y la aplicación se encarga de mostrar dichos mensajes de error al usuario.

## Anexo #3, Construcción de pruebas por componentes para el API

Lo primero que se implementó para realizar las pruebas, fueron los repositorios *mock*, su generación se delegó a una clase *TestHelper.cs* para organizar mejor el código y se puede ver un ejemplo de cómo se construye el *mock* del repositorio para historias clínicas nutricionales en la figura 9.10. Para complementar la imagen, la función *Setup* propia de la clase *Mock* es para establecer qué datos falsos se van a retornar cuando se consulte determinado método del repositorio que se esta simulando.

```
2 references
public static IFormsRepository GetFormsRepositoryMock()
{
  var mockRepository = new Mock<IFormsRepository>();
  mockRepository.Setup(g => g.GenerateForm(Mock.Of<NaForm>()))
    .Returns(Task.FromResult(GetNaFormData()));
  mockRepository.Setup(g => g.GetFormById(It.IsAny<string>()))
    .Returns(Task.FromResult(GetNaFormData()));
  return mockRepository.Object;
}
```

Figura 9.10: Construcción de un repositorio *mock*.

Lo siguiente es montar un client *mock* para poder realizar peticiones de prueba al API sin necesidad de correr la implementación real. Aquí se crea un servidor que incluya todas las instancias y todos los servicios necesarios para las pruebas que usen ese cliente, se ilustra mejor en la figura 9.11.

```
1 reference
private static HttpClient GetClientMock()
{
  var hostBuilder = new WebHostBuilder()
    .UseStartup<Startup>()
    .ConfigureServices(services =>
    {
      services.AddSingleton(TestsHelper.GetMongoConnectionMock());
      services.AddSingleton(TestsHelper.GetNaFieldsRepositoryMock());
      services.AddSingleton(TestsHelper.GetNaSectionsRepositoryMock());
      services.AddSingleton(TestsHelper.GetFieldsRepositoryMock());
      services.AddSingleton(TestsHelper.GetSectionsRepositoryMock());
      services.AddSingleton(TestsHelper.GetAidsRepositoryMock());
      services.AddSingleton(TestsHelper.GetFormsRepositoryMock());
      services.AddSingleton(TestsHelper.GetActivitiesRepositoryMock());
      services.AddSingleton<MailController>();
      services.AddSingleton<TeacherEndpointsController>();
      services.AddSingleton<IFieldsService, FieldsService>();
      services.AddSingleton<ISectionsService, SectionsService>();
      services.AddSingleton<IFormsService, FormsService>();
      services.AddSingleton<IAidsService, AidsService>();
      services.AddSingleton<IActivitiesService, ActivitiesService>();
    });
  var server = new TestServer(hostBuilder);
  var client = server.CreateClient();
  client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue("application/json"));
  return client;
}
}
```

Figura 9.11: Construcción un cliente *mock*.

Una vez se tiene el cliente listo, solo se debe realizar la petición que se desee al *endpoint*, en este caso, haciendo referencia a la figura 9.12, se utiliza de ejemplo la petición a un *endpoint* para actualizar una historia clínica nutricional y se puede notar que después de hacer *hit*, se realiza la verificación apropiada para esta prueba en específico sobre la respuesta que retornó el API.

```
[Fact(Timeout = 1000)]
0 references | Run Test | Debug Test
public async Task UpdateForm()
{
  var client = GetClientMock();
  NaForm form = new ("{formId}"){
    Sections = new List<NaSection>(),
    ActivityId = "507f191e810c19729de860ea",
    StudentId = "278f191e110c45729de86il9"
  };
  var json = JsonConvert.SerializeObject(form);
  var stringContent = new StringContent(json, UnicodeEncoding.UTF8, "application/json");
  var response = await client.PutAsync("/api/v1/forms/{formId}", stringContent);
  Assert.True(response.IsSuccessStatusCode);
}
```

Figura 9.12: Prueba para actualizar una hcn.

## Anexo #4, Pasos para publicar las imágenes de Docker

Los pasos escritos en el documento que ayudaron a estandarizar el despliegue y servir como guía en el proceso son los siguientes (los pasos están escritos en inglés):

**To deploy on EC2**:

1. First, if you're not logged in, run: cat /token.txt | docker login https://docker.pkg.github.com -u <github-email>–password-stdin
2. For safety reasons, run: docker-compose down
3. For downloading newer/required image(s), run: docker pull docker.pkg.github.com/veronicatofino/degree-project/image:version
4. Check that new version of the image is pulled
5. Update version of the image(s) in the service container definition in docker-compose.yml and save the file
6. Once all services are updated, for deploying, run: docker-compose up -d

**To build API image and push it**:

1. First, if you're not logged in, run: cat /token.txt | docker login https://docker.pkg.github.com -u <github-email>–password-stdin

2. For building the image, go to Dockerfile level directory and run:

   docker build -t docker.pkg.github.com/veronicatofino/degree-project/api:version .

3. For pushing the image to registry, run: docker push docker.pkg.github.com/veronicatofino/degree-project/api:version

**To build UI image and push it**:

1. First, if you're not logged in, run: cat /token.txt | docker login https://docker.pkg.github.com -u <github-email>–password-stdin

2. For building the image, go to Dockerfile level directory and run:

   docker build -t docker.pkg.github.com/veronicatofino/degree-project/ui:version .

3. For pushing the image to registry, run: docker push docker.pkg.github.com/veronicatofino/degree-project/ui:version

## Anexo #5, Ejemplo de una actividad para desarrollar

A continuación, se muestra el ejemplo del desarrollo de una actividad por parte del estudiante:

1. En la figura 9.13 se muestra la vista de todas las actividades que tiene un estudiante sin entregar:

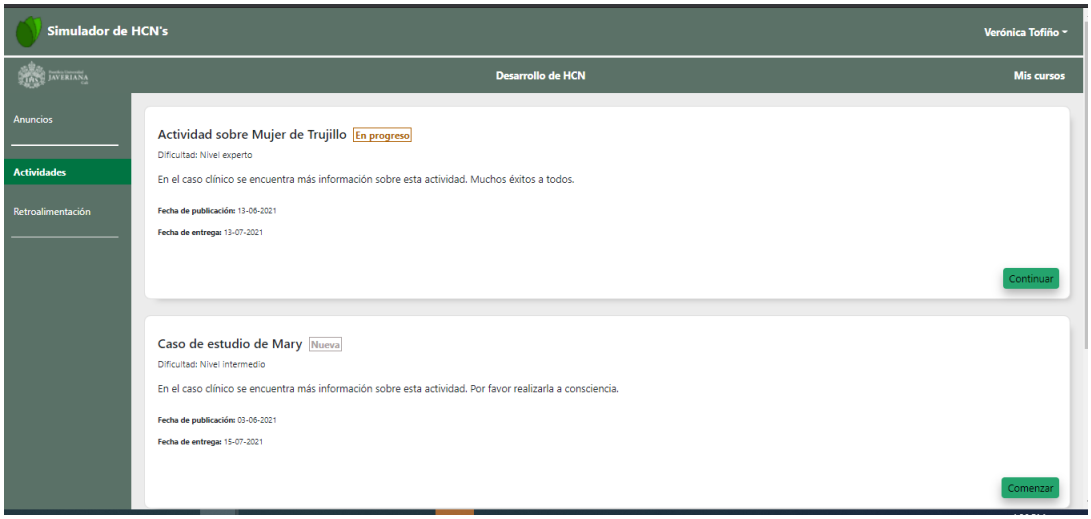



Figura 9.13: Vista de todas las actividades.

Como se ve en la figura anterior, el estudiante tiene dos actividades una en nivel experto y En progreso y otra actividad en nivel Intermedio y sin empezar (Nueva), estas etiquetas ayudan a dar visibilidad al estudiante cuales actividades ya empezó y cuales no, por si se le olvida.

2. Luego, como se ve en la figura 9.14, se le dice al estudiante para una actividad nueva se muestra un modal antes de pasar a la actividad para recordar que el sistema no guarda las actividades de forma automática por lo que debe de hacerlo manual o sino su progreso se perderá. Este modal sólo se muestra si la actividad es nueva.

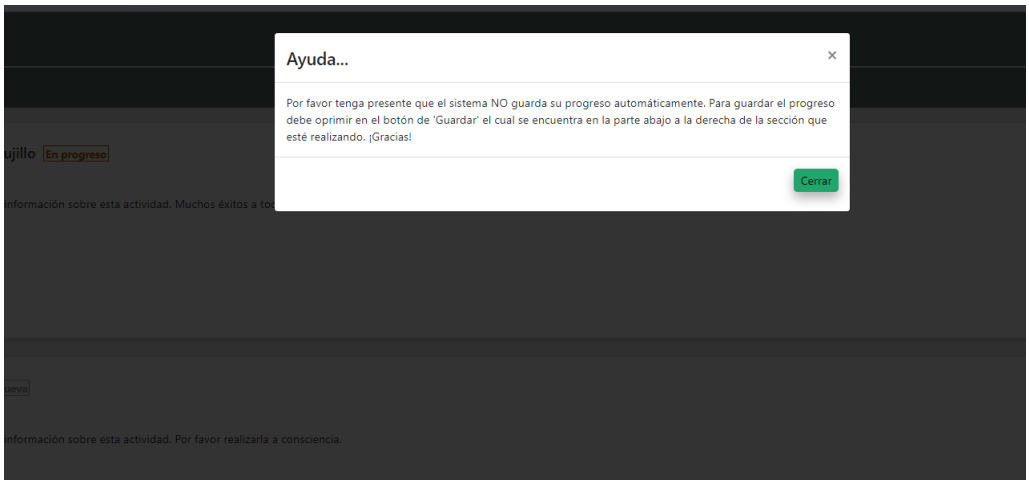



Figura 9.14: Recordatorio actividad nueva

3. Ya en la actividad (figura 9.14) se puede ver la actividad a desarrollar:

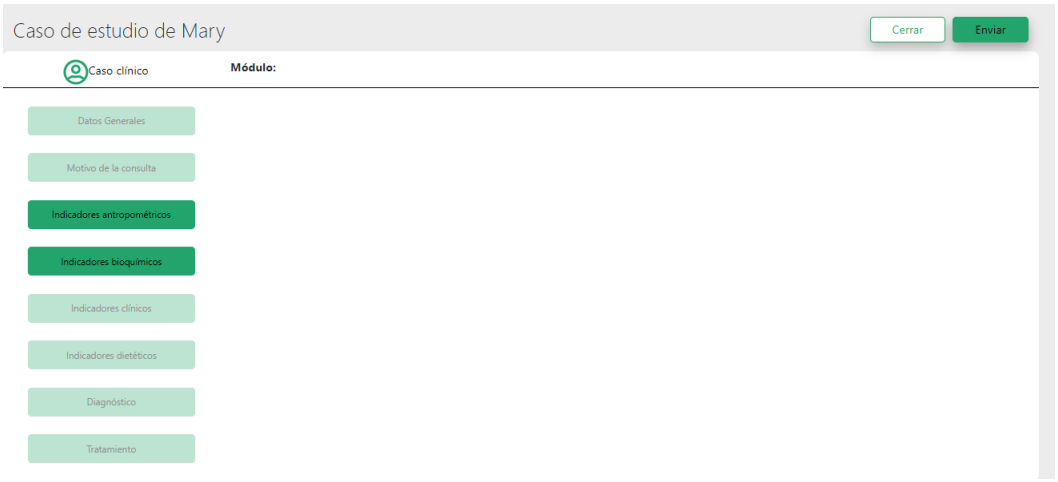
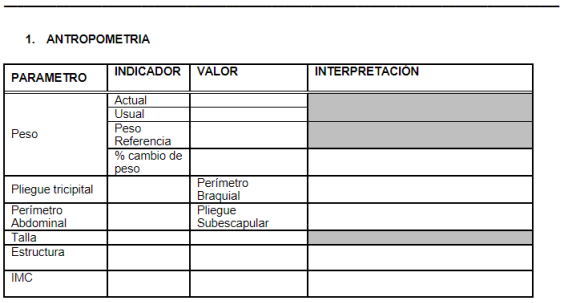



Figura 9.15: Actividad a desarrollar

En la parte izquierda se puede ver un menú con las secciones de una historia clínica nutricional. Sin embargo, se puede ver que para esta actividad solo están habilitadas dos secciones (antropometría y bioquímica) esto es porque las actividades no siempre vienen con todas las secciones para desarrollar. El docente puede escoger que secciones de la HCN quiere que el

estudiante resuelva, por lo que el sistema con base en esos datos habilita sólo las secciones configuradas para la actividad.

4. Al seleccionar, por ejemplo, la sección de antropometría el estudiante visualiza los campos de la sección en forma de tabla (parecido al formato HCN dado por la profesora Lenis). En la figura 9.16 se muestra el formato inicial en Word pasado por la docente y en la figura 9.17 se muestra el desarrollo implementado.



1. ANTROPOMETRIA

| PARAMETRO | INDICADOR | VALOR | INTERPRETACIÓN |
|---|---|---|---|
| Peso | Actual | | |
| | Usual | | |
| | Peso Referencia | | |
| | % cambio de peso | | |
| Pliegue tricipital | | Perímetro Braquial | |
| Perímetro Abdominal | | Pliegue Subescapular | |
| Talla | | | |
| Estructura | | | |
| IMC | | | |

Figura 9.16: Sección de antropometría, formato en archivo Word

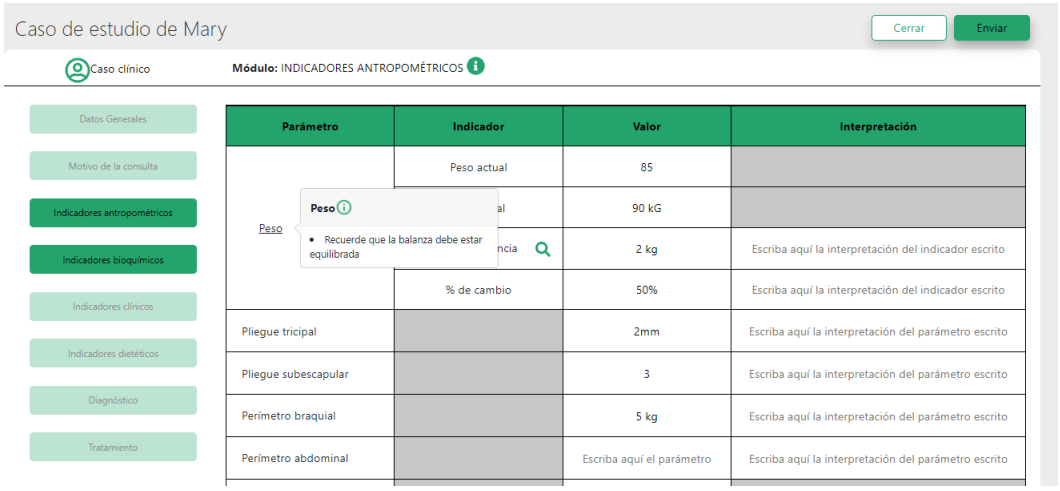



Figura 9.17: Actividad, nivel intermedio, sección antropometría

En la figura 9.17 también se puede ver uno de los tipos de ayuda que ofrece el sistema, la cual es la ayuda de texto flotante. Por ejemplo, en este caso se muestra una ayuda para el campo de "Peso", esta ayuda está configurada para mostrarse en el nivel intermedio, pero puede que para otro nivel no esté configurada y no se muestre. De igual forma si se selecciona la opción de Caso clínico, se abrirá una nueva pestaña mostrando el caso clínico asociado a la evidencia.

5. Por último, luego de enviar la actividad para ser revisada por el docente pasan dos cosas:

la primera es que si el envío fue exitoso al estudiante le llega una notificación a su correo registrado diciendo la fecha de envío y confirmando que fue exitoso (figura 9.18), también le muestra al estudiante toda la actividad que llenó y le da la retroalimentación final del sistema la cual consiste en validar si los campos están vacíos o la medida del campo, fecha, nombre u otros datos conocidos están mal escritos se muestra un icono de error al lado del campo, donde si el estudiante se para le muestra el mensaje diciendo cuál es el error (figura 9.19).

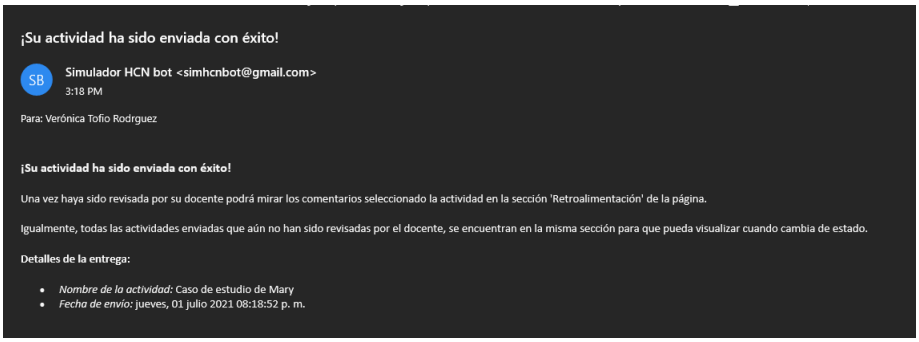



Figura 9.18: Notificación de envío exitoso

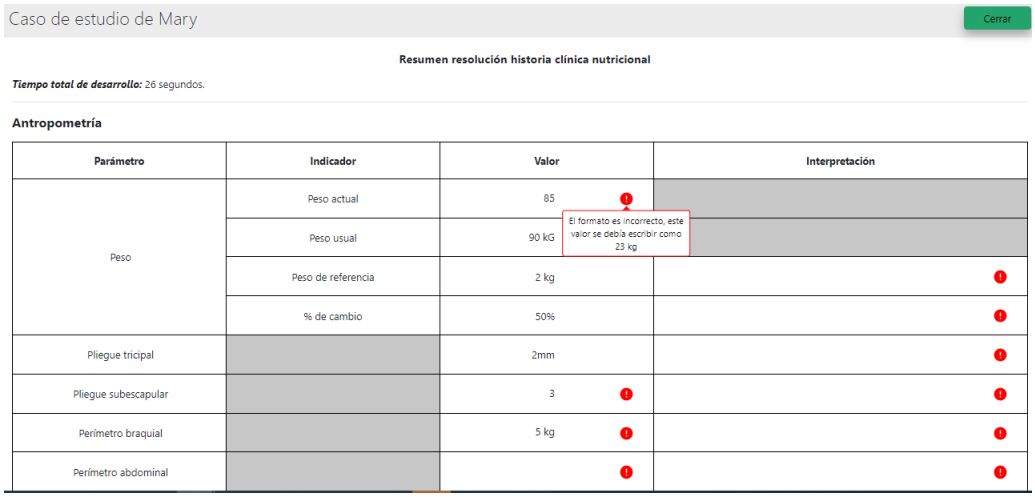



Figura 9.19: Retroalimentación final del sistema

Para mostrar un poco la comparación entre los niveles se muestra primero, en la figura 9.20 la interfaz de desarrollo de actividad, la cual es completamente diferente a la de los otros niveles, también se muestra la retroalimentación inmediata la cual se le da al estudiante después de seleccionar la opción de guardar la sección.

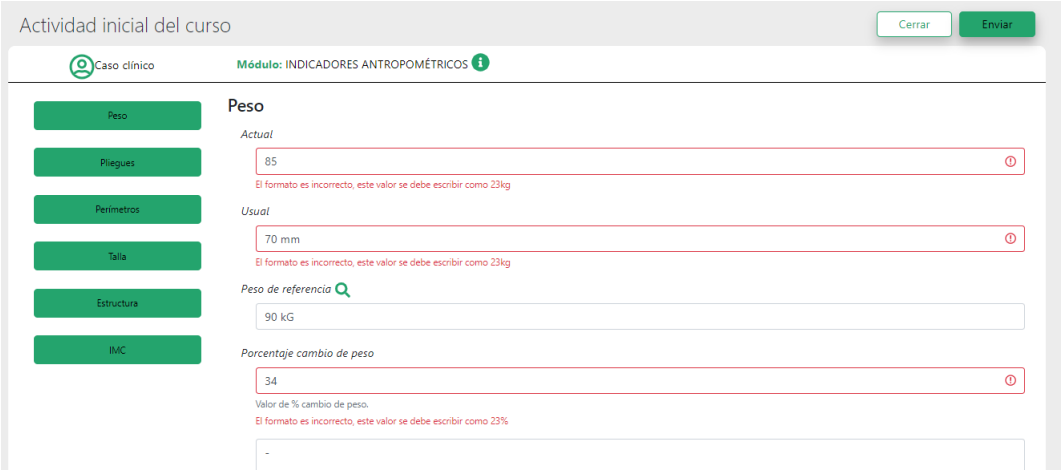



Figura 9.20: Actividad en nivel básico, sección de antropometría

También en la figura 9.21 se muestra un ejemplo de cuales son las ayudas desplegadas para el mismo campo en los niveles básico, intermedio y experto respectivamente.

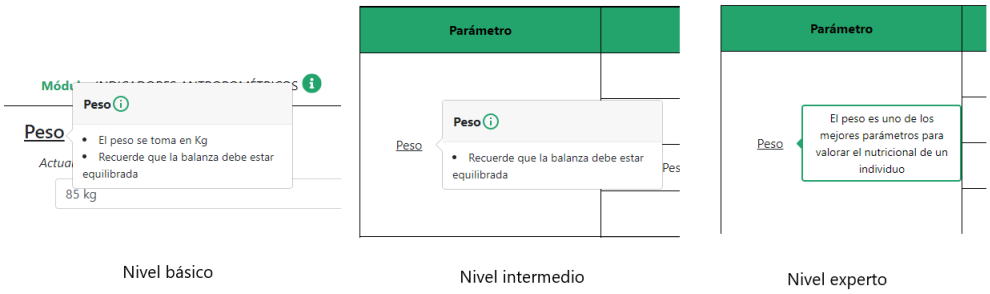



Figura 9.21: Retroalimentación final del sistema