UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Plataforma web para la gestión y asignación de salas bajo el contexto de la Universidad de Talca, Campus Curicó

VICTOR ALEJANDRO REYES MEDINA

Profesor Guía: DANIEL ANTONIO MORENO CÓRDOVA

Memoria para optar al título de
Ingeniero Civil en Computación

Curicó – Chile
Abril, 2020

CONSTANCIA

La Dirección del Sistema de Bibliotecas a través de su encargado Biblioteca Campus Curicó certifica que el autor del siguiente trabajo de titulación ha firmado su autorización para la reproducción en forma total o parcial e ilimitada del mismo.

Curicó, 2022

Dedicado a Cristina Medina, mi madre.

AGRADECIMIENTOS

A mi madre Cristina Medina Hernández, por ser un pilar fundamental durante todo mi desarrollo personal y académico. Contar con su apoyo incondicional en cada etapa de mi vida me ha ayudado enormemente a ser quién soy el día de hoy y a definir quién quiero ser el día de mañana.

A mi padre Victor Reyes González, su preocupación por siempre entregarnos lo mejor posible, ser un ejemplo a seguir y ser un pilar fundamental en mi vida. Desde muy temprana edad he aprendido de él a ser una persona correcta y a hacer las cosas de la mejor forma posible.

A gran parte de mi familia, por su constante apoyo y preocupación. Con especial mención a mis tíos/as Luis Hernández, Ema Medina, Daniel Medina, Andrés Medina y Raquel Reyes. Destacando su apoyo cuando mi salud más lo necesitaba, sin ellos quizás hoy no estaría en este lugar.

A mis amigos Bryan Zuñiga, Daniel Pavez, Matias Erenchun, Pedro González e Ingrid Morales, por ser compañeros en el camino y por ser un soporte vital en estos seis años. De ellos aprendí que se vive mejor con una sonrisa en el rostro y que jamás estarás solo.

A mi profesor guía Daniel Moreno, por su apoyo, confianza y dedicación y por ser un profesor más *humano*. De él aprendí que no importa si viajas largas horas para ir a tu trabajo si estás haciendo lo que realmente te gusta. Gracias por su apoyo.

A los profesores Federico Meza, Rodrigo Paredes, Luis Silvestre y Rodrigo Pavez, por ser profesionales y profesores de calidad que demostraron constantemente su pasión por su trabajo.

TABLA DE CONTENIDOS

página

ÍNDICE DE FIGURAS

página

ÍNDICE DE TABLAS

página

ÍNDICE DE ALGORITMOS

página

RESUMEN

La asignación de salas como un problema de asignación de recursos comúnes, se ve complejizada cuando la cantidad de usuarios solicitantes es mayor que la cantidad de recursos disponibles. En el contexto universitario, la cantidad de recursos (salas) varía solamente cuando se construyen o destruyen nuevos edificios/salas; por su parte la de los usuarios solicitantes (secciones de módulos) varía dependiendo de la cantidad de alumnos y módulos a dictar en un semestre.

Actualmente, el manejo de esta situación reside en el uso de una plantilla Excel y la experiencia de una única persona. Lamentablemente, todos los sub-problemas asociados pueden hacer que todo el proceso de asignación de salas se dificulte y/o extienda demasiado en el tiempo (entre dos y cuatro semanas, habitualmente).

Ante esto, se crea la necesidad de un nuevo sistema o plataforma que ayude a agilizar los procesos tanto en la asignación y gestión de solicitudes como en la consulta de salas disponibles.

En base a lo anterior, se propone un sistema web capaz de gestionar las salas, módulos, secciones y profesores de la universidad. Además, permite realizar asignaciones y gestionar solicitudes de las mismas. Asimismo, el sistema provee una funcionalidad que permite realizar una asignación parcial "semi-automática" de los grupos de bloques de módulos en tan solo un par de minutos, permitiendo posteriormente realizar el resto de asignaciones de forma manual.

La plataforma se desarrolla utilizando Scrum como *framework* de desarrollo. Se detalla el avance logrado en cada iteración indicando la historia de usuario que se completa con la finalidad de que el lector pueda percibir claramente el desarrollo incremental que se obtuvo al aplicar la metodología.

Se especifican dentro de este documento detalles claves para el desarrollo del sistema, tales como la arquitectura física, lógica y el modelo de objetos que lo soportan.

Para finalizar, la validación de la plataforma se realiza usando pruebas de caja negra para comprobar la completitud de las historias de usuario, además de una prueba de usabilidad SUS para identificar las apreciaciones del usuario respecto al sistema.

1. Introducción

El presente capítulo está orientado a describir las bases que llevan a la definición del trabajo a realizar para esta memoria. Primero se describe el contexto en el cual se han detectados los problemas a abordar, luego se definen los objetivos a cubrir, la propuesta de proyecto y finalmente los alcances del desarrollo.

1.1. Descripción del contexto

Cuando hablamos sobre la asignación de recursos comúnes (conocidos como *Common-pool Resources, CPRs* [1]), es imposible no considerar la dificultad inherente que conlleva su uso común o asignación de uso de los bienes. Esto, por que no sólo se decide que cierto recurso es usado por un individuo específico en un determinado tiempo y lugar, sino que se reduce la posibilidad de que otro individuo haga uso efectivo de dicho recurso.

El caso abordado en este documento hace referencia a un problema clásico de asignación de bienes de uso común, la asignación de recursos y se enfoca en un contexto bastante específico: la asignación de salas dentro del Campus Curicó de la Universidad de Talca.

Si consideramos el contexto global en lo que se refiere a asignación de recursos físicos (aulas, canchas deportivas, etc.), usualmente este se realiza con herramientas muy potentes y flexibles pero no necesariamente dedicadas a este contexto en específico (como es el caso del campus Curicó). Por ejemplo, Microsoft Excel, el cual se utiliza para llevar un control y registro tanto de las asignaciones de salas como la definición de horarios, porque a pesar de no ser adecuado, es "fácil de usar" (aunque no necesariamente "usable" en términos de ingeniería de software). También es posible encontrar software específico por el cual se puede pagar

mediante una suscripción mensual o anual; del cual se habla más adelante en el apartado sobre el trabajo relacionado.

Al día de hoy, particularmente dentro del campus Curicó de la Universidad de Talca (en adelante referido como *el campus*), todo este procedimiento es gestionado por sólo una única persona, - a Octubre del 2019 - la Asistente de la Escuela de Ing. Civil Eléctrica, Ninett Vergara.

En aspectos generales, la asignación de salas en el campus consta de dos etapas:

1. Los directores/as de carrera y encargados/as de departamento realizan una planificación semestral del horarios de cada uno de los ramos que se dictarán durante el semestre. Esta información es transcrita en una planilla Excel con un determinado formato, la cual es enviada a la encargada.

2. Una vez se han remitido todas las planificaciones, la encargada debe realizar una asignación manual, en la cual se debe coincidir tanto las capacidades de las salas con los cupos de los ramos como los bloques contiguos de cada uno de los ramos.

Un punto importante a destacar en este proceso es que su segunda etapa es altamente extenuante dada la gran cantidad de información a manejar y los pocos recursos administrados por sólo una persona. Esta actividad puede tomar incluso más de un mes de trabajo dedicado.

Por otro lado, es habitual que existan ciertos eventos académicos que también requieren de la asignación de una sala durante todo el semestre o sólo durante fechas específicas, como por ejemplo las ayudantías y talleres extracurriculares. Otro ejemplo son las de clases de recuperación o pruebas fuera de los horarios habituales de clases, donde un profesor suele solicitar aulas en un determinado bloque, pero sólo por un tiempo limitado o fechas específicas.

Todos estos factores deben ser considerados durante cada semestre al momento de definir un horario y realizar la asignación de salas. En algunos casos, la experiencia previa de los involucrados puede ayudar a definir cierta "estructura" o metodología para realizar las asignaciones. Sin embargo, incluso la ayuda más ínfima (como la posibilidad de ver fácilmente las salas disponibles en un determinado horario), sería altamente valorada.

1.2. Planteamiento del problema

En esta sección se describen las necesidades identificadas que ayudan a definir la solución a entregar.

1. La necesidad de un sistema que permita gestionar el proceso de asignación de salas según su capacidad versus el cupo de cada uno de los ramos y las diferentes secciones dictados en el campus al inicio de cada semestre.

2. La necesidad de un sistema, abierto a la comunidad universitaria, que permita consultar fácilmente las salas disponibles en determinados bloques y poder realizar solicitudes de asignación para una determinada sala en un determinado bloque disponible. Actualmente el proceso de solicitud se realiza mediante correos, llamadas telefónicas o consultas personales a la encargada de la gestión de salas. Por ende, puede no tenerse una respuesta inmediata, o convertirse en una suerte de *negociación* para encontrar una sala disponible en un bloque conveniente.

1.3. Objetivos

En esta sección se definen tanto el objetivo general del proyecto como los objetivos específicos.

1.3.1. Objetivo general

Reducir el tiempo dedicado al proceso de asignación de salas en el campus Curicó de la Universidad de Talca, proveyendo herramientas computacionales que permitan gestionar dicho proceso, mostrando información relativa a las salas, y realizando recomendaciones básicas de asignación.

1.3.2. Objetivos específicos

1. Facilitar la asignación de salas a módulos mediante un sistema web que permita consultar fácilmente las salas disponibles, los bloques a asignar, etc.

2. Facilitar la consulta y asignación de salas para actividades extraordinarias, permitiendo realizar consultas al sistema sobre la disponibilidad de salas en un horario determinado.

3. Facilitar gestión de solicitudes de cambios y/o asignación de salas.

1.4. Propuesta de solución

Se considera el desarrollo de una plataforma web que apoye el proceso de asignación de salas.

En primer lugar, se permite que los directores/as de cada carrera ingresen su planificación del semestre a la plataforma, indicando tanto el nombre del módulo, como su cupo esperado.

El software permite al encargado/a de realizar la asignación de salas realizar una asignación semi-automática de salas, basada en la capacidad propia de cada sala y los requerimientos de espacio de cada uno de los módulos y secciones. Cabe destacar que esta asignación semi-automática puede ser revisada y modificada posteriormente por el encargado/a de forma manual.

El encargado también puede realizar modificaciones a las asignaciones realizadas quitando módulos/ramos y asignándolos a otras salas con diferentes capacidades, a criterio suyo.

Además, se le permite al encargado realizar una asignación totalmente manual en la que puede realizar las siguientes acciones:

- Elegir una sección, ver su horario semanal y asignar una sala en un determinado bloque o rango de bloques.

- Revisar las salas con bloques disponibles para un bloque o grupo de bloques.

- Comprobar el estado actual de una sala, viendo el horario de la misma, con sus bloques disponibles y el ramo que está asignado en los bloques no disponibles.

Además, se permite ver y exportar un reporte del horario semanal por cada carrera con sus módulos y salas asignadas.

Finalmente, el sistema debe permitir a usuarios externos consultar las salas disponibles en bloques determinados. Eventualmente, se puede realizar una solicitud de asignación para una sala en un determinado bloque, la cual debe ser revisada y aceptada por el encargado/a.

Cabe destacar que la solución debe diseñarse con un modelo de datos versátil el cuál permita consultas rápidas además de permitir la escalabilidad y así convertirse en un sistema agnóstico de la institución o contexto que lo esté usando.

1.5. Alcance del proyecto

Dentro del desarrollo de este proyecto se han definido los siguiente alcances:

- Este trabajo se limita a ser desarrollado en base al contexto aplicado dentro de la Universidad de Talca, Campus Curicó. A pesar de ello, pretende ser escalable, para en algún futuro poder ser ampliado a cualquier contexto.
- Para el desarrollo de este trabajo no se considera términos de seguridad en cuanto a la implementación en el servidor.
- El sistema permite hacer una asignación semi-automática de salas, teniendo en consideración que existen casos específicos que -una vez completado el proceso- no tengan una sala asignada. Esto se debe a la falta de recursos físicos (salas) existentes dentro del Campus.

2. Marco teórico

El siguiente capítulo está enfocado en proveer al lector los conocimientos necesarios para comprender cada uno de los conceptos referentes al desarrollo de la plataforma, así como el trabajo relacionado e información sobre las tecnologías usadas durante el mismo.

2.1. Conceptos básicos y trabajo relacionado

Esta sección define conceptos básicos necesarios para comprender el contexto del desarrollo del proyecto.

2.1.1. Conceptos básicos

- **API RESTful**: Se define como una *Application Program Interface* (API) que usa llamadas (o *requests*) HTTP para entregar y recibir información a través de la internet. Está basada en la tecnología *Representational State Transfer*, un estilo arquitectónico y un enfoque de las comunicaciones usado a menudo en el desarrollo de servicios web [2].

- **Framework**: Es un conjunto particular de reglas, ideas o creencias que son usadas para resolver problemas o decidir un curso de acción [3].

- **Single Page Application (SPA)**: Es una aplicación web o sitio web que interactúa con el usuario reescribiendo la página actual (o actualizando parte de ella) en vez de recargar páginas enteras [4].

- **Object-relational mapping (ORM)**: Se trata de una técnica de programación en la que un descriptor de metadatos es usado para conectar el *object code* a una base de datos relacional. El *object code* es escrito en lenguajes orientados a objetos como Java y C# [5].
- **ACID-compliant**: Dígase de un gestor de base de datos tanto relacionales como no relacionales que cumple con los aspectos: Atomicidad (*Atomicity*), Consistencia (*Consistency*), Aislamiento (*Isolation*), Durabilidad (*Durability*) [6].

2.1.2. Trabajo relacionado

Con respecto a software encontrado se encuentran las siguientes propuestas.

1. **Classroom Bookings** [7]. Un sistema open source con opción de suscripción anual. Pese a tener una interfaz simple, cuenta con pocas funcionalidades y además es poco intuitivo. Esto se debe fundamentalmente a que su *demo* no cuenta con una ayuda clara.

   Es bastante personalizable en aspectos como la definición de bloques de horario, aunque su capacidad de personalización puede ser tediosa, debido a las falencias en cuanto a la usabilidad del sistema.

   Por otro lado, no cuenta con el concepto de "asignación de sala a un ramo", ya que las asignaciones permitidas son a un único usuario. Así como tampoco cuenta con la opción para definir en qué bloques de horario es necesario que un "usuario" necesite de una asignación de recursos.

2. **Room Booking System** [8]. Un sistema británico de subscripción anual en base a la cantidad de recursos y cuentas de *staff*. Esto lo hace poco flexible en contextos como el expuesto en este trabajo.

   También cuenta con una gran cantidad de opciones configurables que puede llegar a ser confusa.

   El sistema en sí, no cuenta con opciones para definir un ramo y su horario, simplemente permite definir en qué bloques una sala no está disponible y asignar un "código de clase". Tampoco es posible definir la capacidad de

una sala que permita ayudar a discernir si una sala es apta o no para una clase.

3. **Visual Classroom scheduler** [9]. Un sistema de pago que ha sido diseñado para la administración de las salas. Este sistema es bastante antiguo y cuenta con muchas funcionalidades extras que pueden ser innecesarias. Uno de sus problemas al día de hoy es la compatibilidad con los sistemas actuales.

4. **Design and Implementation of a Classroom Allocation System Prototype** [10]. El cual diseña e implementa un sistema para la asignación automática de salas de dicha universidad.

   Se trata de un sistema desarrollado específicamente para su universidad (Texas A&M University-Corpus Christi), basado en los procesos internos que se usan en la misma. Pese a ser un trabajo para un contexto muy específico, puede ser considerado como un primer avance en cuanto al desarrollo de este tipo de sistemas *a medida*.

   Se basa en leer archivos de entrada en formato Excel, cuya información es utilizada para ejecutar el algoritmo encargado de hacer la comparación de la capacidad de las salas con los requerimientos de cada módulo.

5. **Google Calendar y aplicaciones similares**: En el mercado existen una gran variedad de aplicaciones y/o plataformas que ofrecen funcionalidades orientadas a la organización del tiempo y/o tareas. Sin embargo, estos sistemas no cuentan con el concepto de "recurso asignable por bloques" lo que hace que no se ajusten a los requerimientos en el contexto del campus.

6. **Microsoft Excel**: Es la herramienta actualmente usada en el Campus. Se basa en una plantilla que es rellenada con información de los horarios de cada sección/ramo.

   La versatilidad de Excel hace que el sistema actual se ajuste a las necesidades básicas del Campus. Su interfaz basada en texto y celdas hace que la herramienta sea mínimamente "usable".

El *estado del arte* presentado indica que pese a existir plataformas en el mercado, estas no cubren a cabalidad la necesidad real o se escapan del contexto que

se pretende cubrir dentro de este proyecto; poder gestionar los horarios de los módulos a dictar y asignar salas a los mismos.

Por ejemplo, un sistema de subscripción en base a la cantidad de usuarios y recursos (como el mencionado en el punto 2) es poco viable de implementar en nuestro campus. Ya que usualmente estos planes están basados en cantidad de usuarios o cantidad de recursos a manejar, lo que hace que sean poco viables en contextos cuyos requerimientos no se ajusten a lo ofrecido por las plataformas.

Teniendo en cuenta que al día de hoy, aún no se ha implementado ningún sistema de este tipo en el campus (pese a que se han demostrado intenciones de ello durante bastante tiempo), es válido asumir que la implementación de un sistema de estas características que se ajuste al contexto específico del campus puede aportar en gran medida, no solamente a reducir la carga de trabajo que implica la asignación de módulos sino, que también sería un gran aporte a la comunidad universitaria.

2.2. Tecnologías utilizadas

En la presente sección se describen las tecnologías usadas durante el desarrollo y una breve justificación del porqué de su elección.

2.2.1. Contexto

Como se ha mencionado previamente, el sistema es una plataforma web. Un sistema de tales características debe estar compuesto de al menos 3 elementos, una arquitectura de 3 capas [11].

1. **Capa de presentación (Cliente)**: Aplicación o parte visual del sistema que es accesible por el usuario final. Es accesible a través de internet mediante navegadores web, tanto en computadores como dispositivos móviles [11].

2. **Capa de aplicación (o negocio)**: Programa que se aloja y ejecuta en un computador remoto (usualmente llamado *servidor* o *servidor web*) que computa, gestiona y entrega información a la **Capa de presentación** (obtenida de la **Capa de datos**) comúnmente a través de una API REST. También, en algunas ocasiones puede hacer funciones de *hosting* para la aplicación del cliente [11] .

3. **Capa de datos**: Incluye las bases de datos y la capa de acceso a los datos. En esta capa se pueden encontrar los gestores de bases de datos como MySQL, Postgresql, mongoDB, etc [11].

2.2.2. Herramientas

Durante el desarrollo se han usado frameworks tanto para la aplicación del cliente web como para el servidor backend. A continuación se detallan las herramientas utilizadas:

Aplicación Cliente: Angular

Se ha optado por desarrollar la aplicación del cliente en el framework para desarrollo web de aplicaciones de una sola página (*SPA*), Angular, un framework Javascript para el desarrollo web basado en Typescript y mantenido por Google. Integrado completamente en el entorno Node.js, es capaz de usar la gran mayoría de paquetes/*plugins* disponibles en el gestor de paquetes Npm. Lo cual le aporta una gran versatilidad tanto en el desarrollo como en las funcionalidades que pueden ser provistas al usuario final [12].

En sus primeras versiones, Angular, era conocido como AngularJS ya que funcionaba completamente en Javascript. A contar de la versión 2 es conocido como Angular 2, Angular 2+ o simplemente Angular. Desde ese punto comienza a hacer uso de Typescript, otro lenguaje de código abierto (desarrollado y mantenido por Microsoft) que puede ser descrito como un "superconjunto" de Javascript debido a que en tiempo de compilación Typescript es transpilado a Javascript. Este último puede ser ejecutado en cualquier navegador web [13] [14] .

¿Por qué preferir un Angular por sobre otro framework basado en Javascript? Angular y Typescript funcionan bajo el patrón MVC *out-of-the-box*. En otras palabras, basta con crear un proyecto Angular vacío para obtener inmediatamente un código altamente estructurado y fácilmente mantenible, lo cual permite al desarrollador enfocarse más en implementar nuevas funcionalidades en vez de estructurar el código como tal. A su vez, Typescript se trata de un lenguaje tipado estáticamente, estructurado y basado en clases el cual aporta en gran medida a evitar errores en tiempo de desarrollo que pueden ser altamente probables en lenguajes como Javascript.

Algunas de las alternativas a Angular como framework de desarrollo para aplicaciones web son las siguientes:

1. React, "una librería en Javascript para construir interfaces de usuario". Desarrollada y mantenida por Facebook es uno de los frameworks para el desarrollo web más usados [15].

2. Vue.js, "un framework progresivo de Javascript" [16]. Esto implica que Vue puede ser incluido en una pequeña parte de un proyecto, en vez de que el mismo dependa completamente de este framework.

Ambas alternativas son altamente utilizadas y conocidas por la comunidad. Sin embargo, no proveen de una arquitectura MVC tan definida como en el caso de Angular. Además, ambos usan Javascript. Este es un lenguaje conocido por dejar abierta la posibilidad de que el desarrollador introduzca errores involuntariamente en el código.

Aplicación Backend: .Net Core 2.2

Se decidió implementar la aplicación backend en el framework .Net Core 2.2, un framework de código abierto creado por .Net Foundation con diseño multiplataforma lo que le permite funcionar en Windows, Linux y macOS, a diferencia de su par .Net Framework. Este último está diseñado para funcionar sólo en ambientes Windows [17] . Provee soporte para lenguajes como C#, C++ y F#, además cuenta con el gestor de paquetes de .Net, NuGet, en el cual se puede encontrar gran cantidad de librerías desarrolladas tanto por Microsoft como por la comunidad.

.Net Core es considerado como uno de los primeros pasos de Microsoft para acercarse al mundo *open-source*. Soporta un gran conjunto de herramientas como *web apps*, *web APIs*, aplicaciones por línea de comandos, librerías y aplicaciones *Universal Windows Platform*.

Para el desarrollo de este proyecto se ha creado una *web API* en .Net Core 2.2 con el lenguaje C# destacando el uso de librerías como Entity Framework Core (ORM) e Identity Framework (gestión de autenticación).

¿Por qué preferir un .Net Core como web API por sobre otro framework basado backend? .Net Core, en este caso específicamente C# como lenguaje estructu-

rado y fuertemente tipado se presenta como una herramienta robusta y confiable para el desarrollo de una aplicación donde el *release* de cada versión es altamente probado y validado antes de su liberación al público.

Algunas de las alternativas más conocidas a .Net Core como framework de desarrollo de aplicaciones de backend son las siguientes:

1. Express.js, framework para el desarrollo backend implementado y ejecutado en el ambiente Node.js."Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles" [18].

2. Java Spring, framework de desarrollo para Java.

3. Lumen, framework minimalista para el lenguje PHP. Orientado al desarrollo de microservicios y APIs.

Servidor de Bases de Datos: Postgresql

Se optó por PostgreSQL como gestor de bases de datos relacional en su última versión. PostgreSQL se trata de un proyecto de código abierto, mantenido por la comunidad *PostgreSQL Global Development Group*.

El hecho de que sea totalmente *ACID-compliant* a una velocidad incluso mejor que MySQL [19] ubica a PostgreSQL como uno de los gestores más conocidos y usados dentro de la comunidad de desarrollo de software y ciencia de datos. Además, su licencia de uso es muy similar las conocidas BSD y MIT lo que otorga una gran versatilidad a los desarrolladores, ya que incluso se permite el uso comercial o en proyectos de código abierto o cerrado [19, 20].

¿Por qué preferir un PostgreSQL como gestor de bases de datos por sobre otro gestor? Uno de los aspectos más importantes que conllevan en uso de PostgreSQL es que puede llevarse a un ambiente de producción sin la necesidad de optar por otro tipo de licenciamiento como en el caso de MySQL [19, 20].

Alternativas

1. MySQL

2. Microsoft SQL Server

2.3. Metodología

Esta sección está orientada a presentar los conceptos básicos sobre la metodología de desarrollo usada en el proyecto.

2.3.1. Metodología de desarrollo: Scrum

Scrum es un *framework* de desarrollo ágil, basado en el desarrollo iterativo e incremental, hace énfasis en tomar decisiones orientadas a responder los cambios en los requerimientos y/o necesidades en vez realizar especulaciones hechas incluso antes de que el desarrollo propiamente tal comience [21].

A continuación se indican los roles que forman parte de la implementación de Scrum como *framework* de desarrollo [22].

- **Product Owner**: Responsable de la visión del producto, la gestión del *Product Backlog* y de considerar los interses de los *Stakeholders*. Tiene la facultad de aceptar o rechazar el crecimiento del producto.
- **Scrum Development Team**: Equipo auto-organizado y multifuncional sin roles asignados o estáticos. Durante cada *Sprint* trabajan colaborativamente para entregar al final de este un producto funcional.
- **Scrum Master**: Encargado de facilitar y cumplir el proceso de *Scrum*. No tiene autoridad sobre la gestión del equipo de desarrollo (por definición, cualquier persona con autoridad sobre el equipo, no es un *Scrum Master*). Se presenta como líder del equipo, atendiendo las dudas e impedimentos del mismo. Actúa como canal conductor para las tareas entregadas por el *Product Owner* hacia el *Development Team*.

A continuación se presentan los eventos o ceremonias que un equipo que implemente el *framework* Scrum *debe* seguir [22].

- **Sprint Planning**: Al inicio de cada *Sprint* el *Product Owner* se reúne con el equipo de desarrollo y negocian cuáles ítems del *Product Backlog* deberán ser abordados para convertir en partes funcionales del producto final. Es responsabilidad del *Product Owner* declarar cuáles ítems son más importantes para el negocio. Por su parte, el equipo es responsable de declarar la cantidad de trabajo que ellos creen pueden realizar durante el *Sprint*.

- **Sprint**: Es el periodo de tiempo definido en el cual el *Development Team* trabaja en conjunto para entregar incrementalmente una nueva funcionalidad al producto.
- **Daily Scrum Meeting**: Al inicio de cada día el *Development Team* se reúne a reportarse mútuamente a cada uno el estado de su trabajo. Cada miembro resume qué hizo el día anterior, qué hará hoy y qué problemas enfrenta.
- **Sprint Review**: Una vez que cada *Sprint* haya finalizado, el equipo se reúne para demostrar y revisar el incremento en el producto funcional en conjunto con el *Produt Owner* y cualquier interesado en el proyecto. La reunión consiste en una demostración, no un reporte.
- **Sprint Retrospective**: Cada *Sprint* termina con una reunión de retrospección. En estas el equipo analiza su propio proceso y comportamiento. Además se decide el curso de acción para mejorar (o detener) los problemas importantes que hayan detectado.
- **Backlog Refinement**: La mayoría de *Product Backlogs* necesita refinarse o mejorarse con el tiempo. Por lo tanto, es importante que el equipo considere cierto tiempo al análisis y mejora del mismo.

A continuación, la Figura 2.1 indica el flujo de ceremonias que ocurre en cada *Sprint* de Scrum.

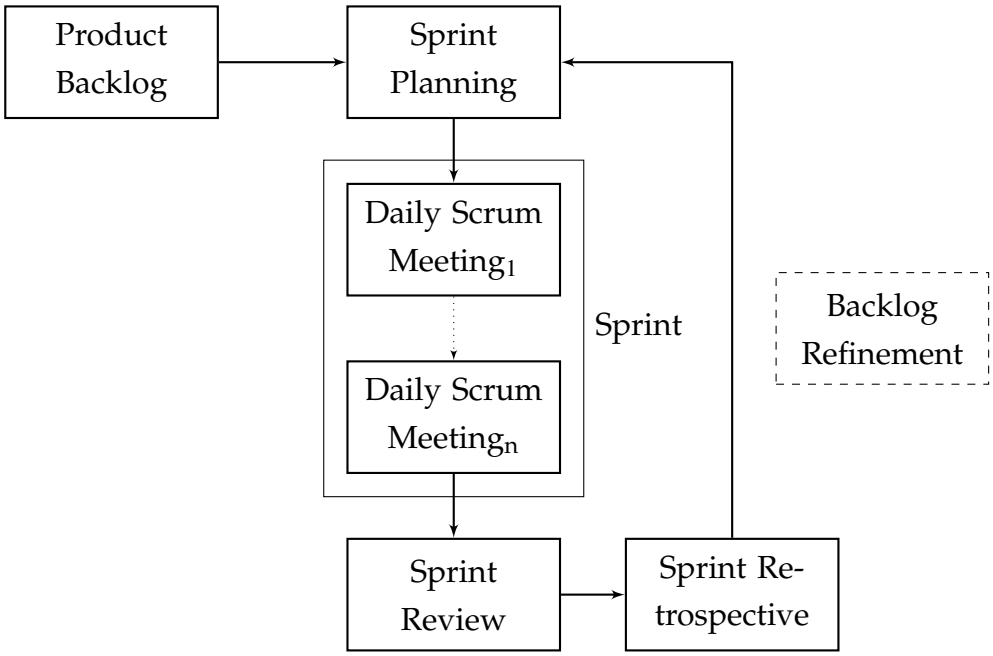


Figura 2.1: Flujo de eventos/ceremonias de Scrum

2.4. Metodología de pruebas

En esta sección se identifica las metodologías a utilizar con respecto a las pruebas que se realizan al sistema. Estas pruebas están enfocadas a demostrar el cumplimiento de los objetivos generales y específicos.

2.4.1. Pruebas de usabilidad (SUS)

El Sistema de escala de usabilidad (SUS, System Usability Scale) es una herramienta confiable y de fácil implementación [23]. Permite evaluar una amplia gama de productos y servicio. Consta de un cuestionario de 10 preguntas, las cuales son evaluadas usando una escala de Likert donde se indica el grado de acuerdo o desacuerdo en cada una de ellas [24]. Con este tipo de pruebas, se pretende demostrar el grado de usabilidad y/o satisfacción del sistema por parte de los usuarios.

El detalle en profundidad de la definición y aplicación de pruebas de usabilidad es tratado en el Capítulo 5.

2.4.2. Pruebas de caja negra

Pruebas de caja negra se refiere a la metodología de pruebas de software que es usada para probar o *testear* software sin saber cómo funciona el programa internamente. De esta forma, es posible verificar si un software *"funciona"* desde el punto de vista de un usuario final del software [25].

La Figura 2.2 ejemplifica el flujo de una prueba de caja negra. En la cual un tester usa un conjunto de entradas y verifica que el conjunto de salidas se valide con el conjunto de resultados esperados.

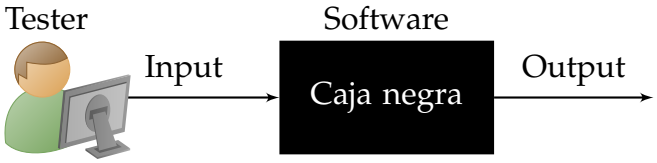


Figura 2.2: Diagrama que representa el flujo de una prueba de caja negra

El detalle en profundidad de la definición y aplicación de las pruebas de caja negra es tratado en el Capítulo 4.

3. Metodología y requisitos

Este capítulo describe cómo se ajusta la metodología al contexto del desarrollo particular de este proyecto, además de la definición de las historias de usuario y su valorización.

3.1. Metodología

Uno de los factores más frecuentes por lo que un proyecto de desarrollo de software puede fallar es la elección errónea de una metodología de trabajo [26]. Por lo tanto, elegir la metodología correcta, o al menos la que más se ajuste a las necesidades, es crucial tanto para la salud del equipo de desarrollo como para la calidad del software en sí.

Se ha optado por el uso de una metodología ágil. Ésta permite una gran flexibilidad y capacidad de respuesta frente a cambios de prioridades o requerimientos. También otorga tener una mayor cercanía con los *stakeholders*.

Además, uno de los puntos importantes que implica una metodología ágil es la entrega continua e incremental de productos funcionales. Esto se ajusta al contexto académico donde se desarrolla el proyecto.

Finalmente, la metodología (o *framework*) que se ha elegido para el desarrollo del proyecto es Scrum, pero esta vez se ha modificado para ser usado por *un equipo de un solo hombre*. Se mantienen algunas ceremonias y se modifican las que requieran la participación de un equipo completo de desarrollo y se determinó la duración de cada *Sprint* en dos semanas.

A modo de control de la ejecución del desarrollo, se hace uso de determinados elementos de la metodología Kanban. Conformando una metodología híbrida que

integra aspectos tanto de Scrum como de Kanban.

La metodología Kanban conlleva el uso de un tablero, el cual provee un *feedback* visual del flujo de trabajo en conjunto con las tareas. La Figura 3.1 ilustra el tablero Kanban usado por la plataforma Azure Devops[1]. Cada una de las tarjetas en blanco puede representar una historia de usuario o tarea, entre otras opciones que permite la plataforma.

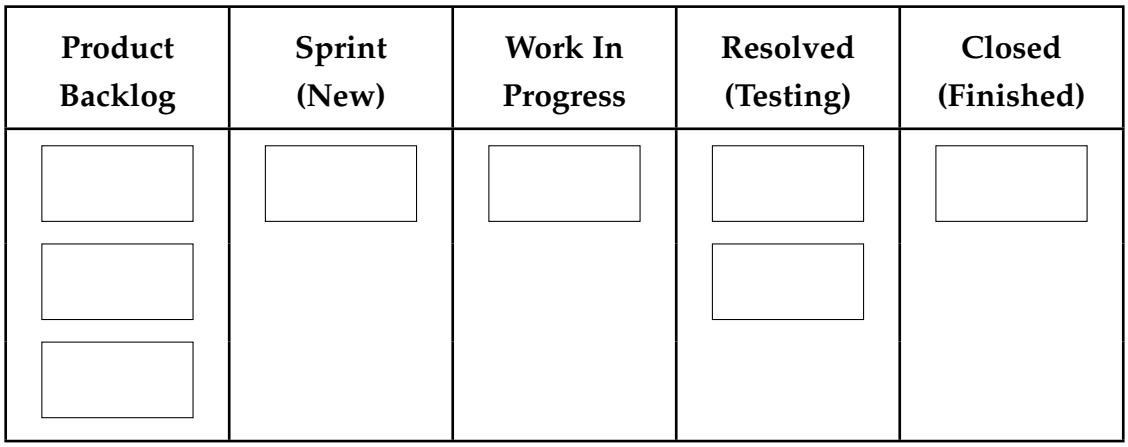


Figura 3.1: Ejemplo de tablero Kanban (basado en tablero usado en Azure Devops)

3.2. Requisitos del sistema

Una de las formas de representar los requerimientos del software es mediante la definición de historias de usuario, las cuales están descritas y definidas en la presente sección.

Las historias de usuario son una herramienta usada en el desarrollo ágil que describe una funcionalidad del software desde el punto de vista de un usuario. Por su parte, una épica del sistema es un gran conjunto de trabajo, en este caso historias de usuario, usado para agrupar con mayor abstracción los requerimientos del sistema.

[1]Disponible en: `https://docs.microsoft.com/en-us/azure/devops/boards/boards/kanban-quickstart?view=azure-devops`

3.2.1. Épicas

En esta sección se definen las Épicas o *epics* del sistema de los cuales se desprenden las historias de usuario.

El código de cada épica está descrito como `Ex`, siendo *x* un número entero, formando así un identificador único para cada épica.

Tabla 3.1: Épicas del sistema

Código	Epic
`E1`	Gestión de carreras y horarios de ramos/módulos.
`E2`	Gestión de edificios y salas.
`E3`	Gestión de asignaciones y solicitudes de asignación y cambios.

3.2.2. Historias de usuario

En esta sección se describen las historias de usuario definidas por cada uno de los usuarios del sistema.

El código de cada historia de usuario está formado por dos partes. Por ejemplo en `Ex-Sy` (siendo *x* e *y* números enteros) la primera parte indica a qué épica pertenece y la segunda la identifica dentro de dicha épica.

Historias de usuario para Director/a de carrera

A continuación, la Tabla 3.2 describe la historias de usuario correspondientes a un director/a de carrera.

Tabla 3.2: Historias de usuario - Director/a de carrera

Código	**Yo, como director/a de carrera quiero...**
`E1-S1`	...Indicar qué ramos con sus respectivas secciones se dictarán en mi carrera durante el semestre, para así poder enviar una solicitud de asignación de salas a todos los ramos definidos.
`E1-S2`	...Indicar en el sistema la cantidad de alumnos estimados para cada módulo/sección inscrito, con la finalidad de que la asignación del recurso sala esté acorde a las necesidades propias del módulo y/o sección.
`E1-S3`	...Asignar un horario para una determinada sección, para definir en cuál/es días y bloques de horario deben ser asignadas las salas para dicho módulo.
`E1-S4`	...Definir qué bloques de horario de un curso no necesitan una sala de clases, para evitar asignar recursos sala innecesariamente.
`E1-S5`	...Identificar qué bloques de horario necesitan usar laboratorios en vez de salas de clases, para que el recurso laboratorio sea asignado de forma particular.
`E1-S6`	...Enviar una solicitud de inscripción de ramos que será gestionada por el encargado con la finalidad de que sea él/ella el o la responsable de hacer una correcta asignación de recursos.

Historias de usuario para Profesor

La Tabla 3.3 define las historias de usuario correspondiente a un profesor del campus.

Tabla 3.3: Historias de usuario - Profesor

Código	**Yo, como profesor de la universidad quiero...**
E3-S1	...Consultar en cualquier momento la existencia de una sala disponible un determinado día de la semana en un bloque/s específico/s, para poder saber rápidamente si es factible (por ejemplo) designar una ayudantía en un determinado bloque del día.
E3-S2	...Crear una solicitud de asignación (*solicitud extraordinaria*) de sala en un determinado día de la semana en un bloque/s específico/s para actividades como ayudantías o talleres, así confirmar un espacio físico donde realizar las determinadas ayudantías o talleres.
E3-S3	...Crear una solicitud de asignación de sala especial (*solicitud especial*), la cual tendrá efecto solamente durante un tiempo determinado para actividades como pruebas fuera del horario de clases o clases de recuperación. Ejemplos de estas situaciones son: "Viernes 26 de abril en el bloque 6 y viernes 31 de mayo en el bloque 6" y "Lunes bloque 8, durante 3 semanas consecutivas". De esta forma, las salas solicitadas volverán a estar disponibles una vez que se cumpla el vencimiento de la asignación.
E3-S4	...Crear una solicitud de cambio de sala y/o horario a conveniencia para un módulo que esté dictando durante el semestre, para así realizar modificaciones en conjunto con los alumnos sobre el horario y sala donde se dicta un ramo.

Historias de usuario para Estudiantes del campus

La Tabla 3.4 define la historia de usuario correspondiente a cualquier estudiante del campus (en este proyecto, solo se considera una historia de usuario de estudiante).

Tabla 3.4: Historias de usuario - Estudiante

Código	Yo, como estudiante de la universidad quiero...
E3-S5	...Acceder a un sistema en el cual pueda ver la disponibilidad de una sala en algún día de la semana en determinado bloque para de está forma encontrar salas disponibles que pueda usar para organizar actividades (por ejemplo, realizar una reunión de equipo donde pueda usar la pizarra para hacer lluvias de ideas).

Historias de usuario para Encargado/a de gestión de salas

La Tabla 3.5 describe las historias de usuario correspondiente al encargado/a de la gestión de salas de la universidad.

Tabla 3.5: Historias de usuario - Encargado/a gestión de salas

Código	Yo, como encargado de la gestión de salas quiero...
E2-S1	...Crear nuevas salas indicando su capacidad para contar con ellas para las siguientes asignaciones de salas.
E2-S2	...Crear nuevos edificios en el sistema y asignar/crear salas en ellos, para así mantener de forma más ordenada las salas de la facultad.
E1-S7	...Crear una nueva carrera que se dicta dentro del campus para poder recibir solicitudes de asignaciones de parte de dicha carrera.
E3-S6	...Recibir todas las solicitudes de asignación de salas de todas las carreras para así centralizar el proceso y que solamente una persona sea encargada de realizar la asignación de salas.

Continuación Tabla 3.5	
Código	**Yo, como encargado de la gestión de salas quiero...**
E3-S7	...Realizar una asignación semi-automática de las salas a los módulos y secciones. Donde posteriormente pueda modificar detalles de la misma. De esta forma, podré realizar más rápidamente todo el proceso de asignación de salas .
E3-S8	...Realizar una asignación manual de las salas a los módulos y secciones. Determinado en cada caso en qué sala se dictará cada módulo/sección. De esta forma, podré realizar asignaciones en base a experiencia propia o preferencias que no podrían ser simuladas en el sistema.
E3-S9	...Gestionar las *solicitudes de extraordinarias* para aceptar o rechazar las mismas.
E3-S10	...Gestionar las *solicitudes especiales* para así aceptarlas o rechazarlas, además de poder visualizar las que están actualmente activas.

3.3. Priorización y estimación de historias de usuario

Durante el proceso de priorización de historias de usuario se ha usado el método conocido como *MoSCoW*. Se trata de un método comúnmente utilizado debido a su facilidad para ser entendido tanto como por analistas como *stakeholders* que no estén directamente relacionados con el desarrollo de software [27].

El método *MoSCoW* define las siguientes categorías.

- **M - Must Have**. Los requerimientos bajo esta categoría deben estar contenidos en el proyecto. Fallar al cumplir estos requisitos significa que el proyecto en su totalidad puede ser un fracaso.
- **S - Should Have**. Requerimientos o funcionalidades de alta prioridad pero sin la necesidad crítica de ser implementados. Agregan valor e importancia

al proyecto por parte del usuario.

- **C - Could Have**. Requerimientos deseables pero no estrictamente necesarios. Su importancia es menor a los contenidos en el grupo **Should Have**.
- **W - Won't Have**. Requisitos que no se incluirán dentro del desarrollo del proyecto actual. Sin embargo, pueden ser considerados en una etapa futura.

Por otro lado, la estimación de cada historia de usuario se ha realizado en base a *Story Points* con una escala arbitraria, pero conveniente, de 1 a 5.

3.3.1. Historias de usuario para Director/a de carrera

La Tabla 3.6 indica la prioridad y estimación de las historias de usuario correspondiente a un director/a de carrera.

Tabla 3.6: Priorización historias de usuario - Director/a de carrera

Código	Priorización	Estimación
E1-S1	Must Have	3 Story Points
E1-S2	Must Have	2 Story Points
E1-S3	Must Have	5 Story Points
E1-S4	Must Have	1 Story Points
E1-S5	Won't Have	2 Story Points
E1-S6	Must Have	4 Story Points

3.3.2. Historias de usuario para Profesor

La Tabla 3.7 define la prioridad y estimación las historias de usuario correspondiente a un profesor del campus.

Tabla 3.7: Priorización de historias de usuario - Profesor

Código	Priorización	Estimación
E3-S1	Must Have	4 Story Points
E3-S2	Must Have	4 Story Points
E3-S3	Must Have	5 Story Points
E3-S4	Could Have	4 Story Points

3.3.3. Historias de usuario para Estudiantes del campus

La Tabla 3.8 determina la prioridad y estimación de las historias de usuario correspondiente a cualquier estudiante del campus.

Tabla 3.8: Priorización de historias de usuario - Estudiante

Código	Priorización	Estimación
E3-S5	Should Have	2 Story Points

3.3.4. Historias de usuario para Encargado/a de gestión de salas

La Tabla 3.9 define la prioridad y estimación de las historias de usuario correspondiente al encargado/a de la gestión de salas de la universidad.

Tabla 3.9: Priorización de historias de usuario - Encargado/a gestión de salas

Código	Priorización	Estimación
E2-S1	Must Have	2 Story Points
E2-S2	Must Have	2 Story Points
E1-S7	Must Have	2 Story Points
E3-S6	Must Have	3 Story Points

Continuación Tabla 3.9		
Código	**Priorización**	**Estimación**
E3-S7	Must Have	5 Story Points
E3-S8	Must Have	5 Story Points
E3-S9	Must Have	4 Story Points
E3-S10	Could Have	4 Story Points

4. Desarrollo

El presente capítulo describe los elementos y/o etapas claves de la fase de desarrollo de software llevada a cabo para la ejecución de este proyecto.

4.1. Diseño

Esta sección describe una de las etapas del ciclo de desarrollo de software más importante; el diseño del software. Esta fase contempla tanto la definición de la arquitectura física y lógica del sistema como el de los componentes visuales (interfaces).

La importancia de la fase de diseño radica en la capacidad y necesidad de contener la propagación de errores que se puedan introducir.

4.1.1. Arquitectura física

A continuación, la Figura 4.1 ilustra la definición de la arquitectura física diseñada.

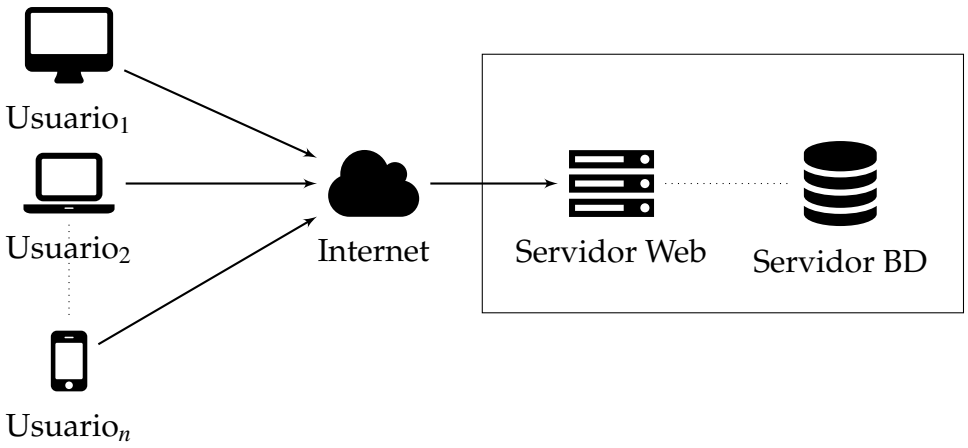


Figura 4.1: Diagrama que representa la arquitectura física del sistema

Se trata de una arquitectura en cliente-servidor en n-capas en su variante de tres capas, donde la primera son los usuarios, la segunda es el servidor web que *aloja* la aplicación y la tercera (de existir) es el servidor de base de datos. Esto último se debe a que en determinados casos el servidor web y el servidor de base de datos pueden estar *hospedados* en el mismo servidor físico, combinando así las capas dos y tres en solamente una segunda capa. Este tipo de variaciones en la arquitectura es transparente para el usuario, es decir, la plataforma funciona de la misma forma sin importar si el servidor web y de base de datos están *alojados* en el mismo servidor físico o no.

Es relevante mencionar que mantener una arquitectura de este tipo ayuda a mejorar pasivamente la seguridad del sistema en general.

Este tipo de arquitectura permite que el sistema se ejecute o *aloje* en un dispositivo independiente del que estén usando los usuarios, el servidor web. Esto permite la inter-operabilidad de la plataforma independientemente del dispositivo o sistema operativo que estén usando los usuarios.

Cabe destacar, que al usarse Angular como *framework* para el desarrollo front-end, la aplicación del cliente es ejecutada en el navegador en el dispositivo del usuario. Esto se debe a que es el servidor web quien se encarga de entregar datos a la aplicación en el cliente así como la aplicación misma.

Por su parte, es solamente el servidor web quién se comunica con el servidor de base de datos para obtener la información.

Al momento de escribir éste documento, la aplicación desarrollada se encuen-

tra en un ambiente de *producción/pruebas* usando una arquitectura cliente-servidor de 2 capas, ya que el servidor web Kestrel[1] se ejecuta desde un contenedor Docker y la base de datos está siendo ejecutada localmente por el servidor. Todo esto ocurre dentro de una instancia de *Google Cloud Platform - Compute Engine* ubicada físicamente en Carolina del Sur, EEUU.

4.1.2. Arquitectura lógica

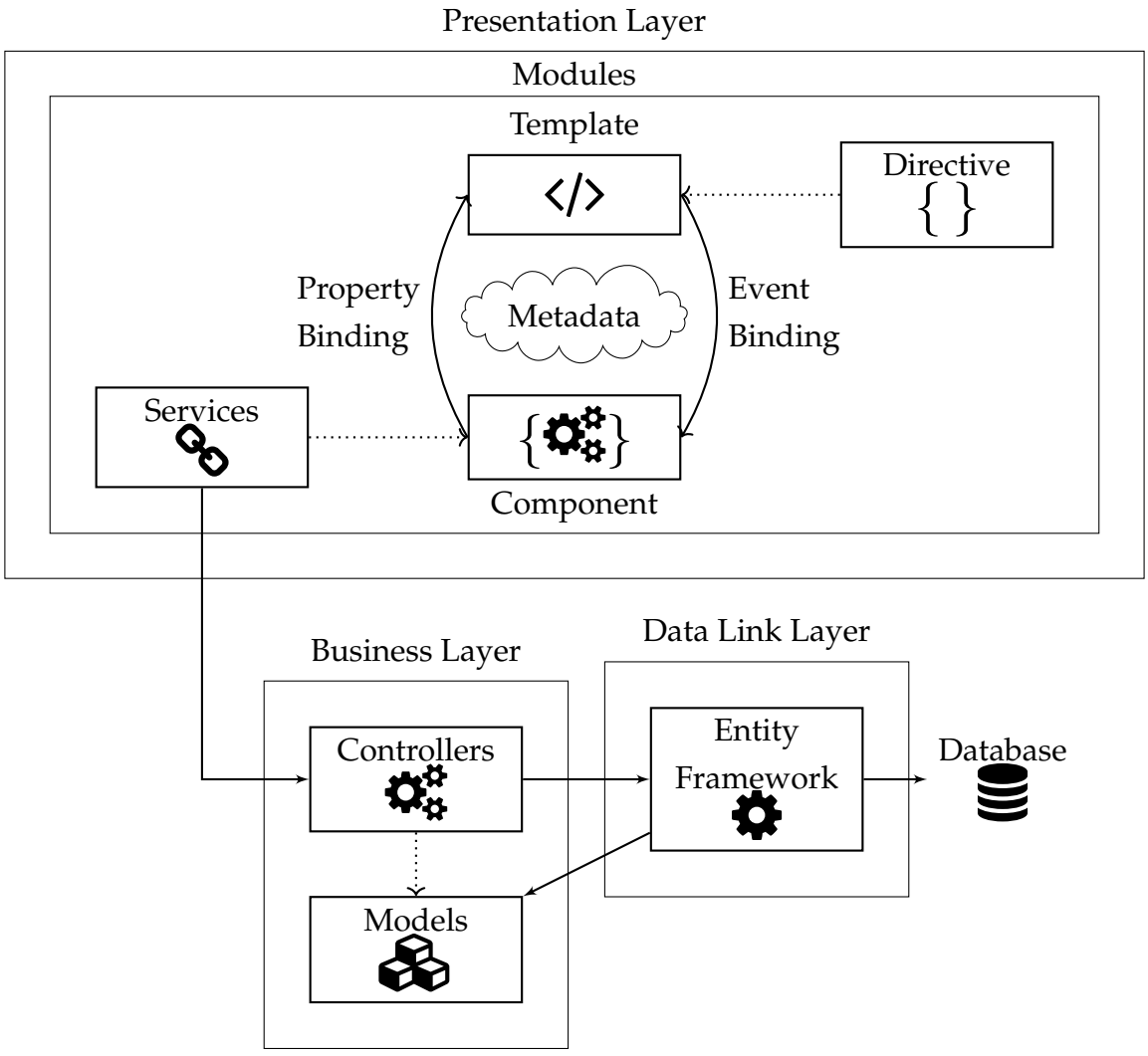


Figura 4.2: Diagrama que representa la arquitectura lógica del sistema

[1]Kestrel: Motor de servidor web multiplataforma para .Net Core. Más información en: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-2.2

La Figura 4.2 describe las tres capas lógicas del sistema. La capa de presentación (*Presentation Layer*) encargada de actuar como interfaz para el usuario, la capa del negocio (*Business Layer*) encargada de modelar ciertas necesidades propias del contexto del problema y la capa de acceso a datos (*Data Link Layer*) que es la encargada de controlar y ejecutar las acciones en la base de datos.

La capa de presentación (cuyo diagrama de arquitectura lógica ha sido simplificado) está desarrollada íntegramente en el *framework* Angular.

La arquitectura de una aplicación Angular está compuesta de módulos (conocidos como `NgModules`) los cuales además de comunicarse entre sí, pueden contener sus propios componentes y servicios. La característica principal a destacar de este *framework* es la relación entre la vista (*Template*) y su controlador (*Component*), que mediante el *Event Binding* y *Property Binding* permite que tanto la vista como el controlador interactúen dinámicamente cada vez que uno de ellos es modificado. El controlador hace uso de dependencias inyectadas (como los servicios *services*) para obtener información, como por ejemplo, hacer una llamada `http` a la API para obtener la lista de secciones de una carrera.

La modularización "nativa" de Angular entre sus módulos y componentes (que simula en gran medida al patrón de diseño Modelo-Vista-Controlador, MVC) hace que cada uno de estos últimos tengan un bajo grado de acoplamiento entre sí y un alto grado de cohesión.

Está demás mencionar que solamente los servicios tiene comunicación con la siguiente capa, la capa del negocio, que al tratarse de una API Rest, la capa del negocio puede ser fácilmente cambiada sin necesidad de modificar la capa de presentación.

Tanto la capa del negocio como la capa del acceso a datos están implementadas en .Net Core. En esta ocasión ambas capas si están altamente acopladas. Esto se debe a una decisión de diseño que se tomó al inicio del proyecto que es abordado más adelante en este mismo capítulo.

En la capa del negocio, son los controladores *Controllers* quienes se encargan de responder ante cada llamada `http`. En cada llamada que requiere de obtener información desde la base de datos el controlador debe acceder mediante el ORM *Entity Framework*. Este último es el único con acceso directo a la base de datos. En este nivel, tanto los controladores como *Entity Framework* hacen uso de los modelos *Models* definidos en la capa del negocio. Estos modelos son una representación

abstracta de contexto del problema.

4.2. Modelo de clases

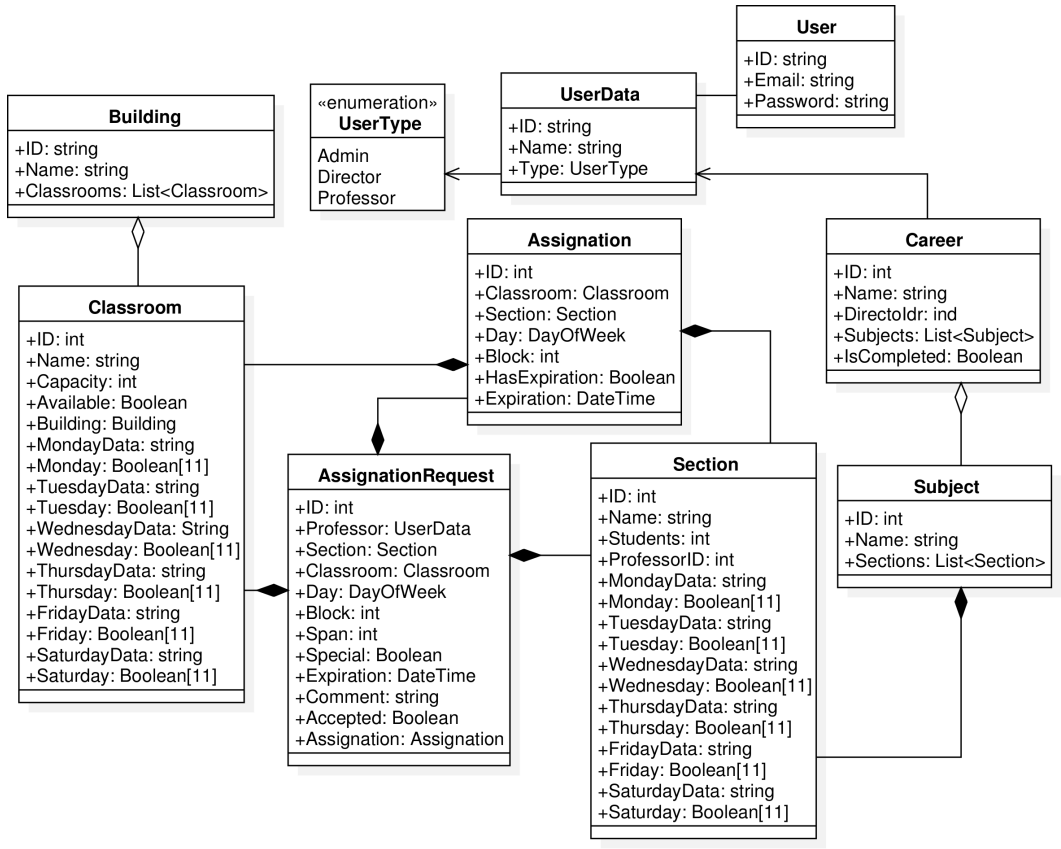


Figura 4.3: Diagrama que representa el modelo de clases

El ORM *Entity Framework* para .Net Core en su modalidad *Code-First*, permite la facilidad de que el modelo de clases implementado por el desarrollador sea usado para crear un modelo relacional en el motor de bases de datos deseado.

Es decir, no existe la necesidad de diseñar ni menos codificar un *script* de base de datos. Ya que el ORM es el encargado de "traducir" de forma transparente para el desarrollador su modelo de clases. Esto también implica que permite acceder a la base de datos como si se trata de una colección (como `List<>`) y así usar ciertas funcionalidades propias de .Net Core y C# para consultas (*querys*) en listas como

lo es LinQ[2].

La Figura 4.3 describe usando notación UML el diseño de clases usando el sistema.

A pesar de que se usó un ORM y que el diseño de la estructura de la base de datos es transparente para el desarrollador, sí se debe diseñar el modelo de clases teniendo en mente el resultado esperado para la base de datos. Esto se aprecia al tomar ciertas decisiones de diseño. Por ejemplo, una de las importante decisiones de diseño realizadas es el no hacer uso de herencia entre las clases `Assignation` y `AssignationRequest`. Esto último se debe a que (en base a las "traducciones" o "mapeo" del ORM) la entidad resultante contendría los datos de ambas clases con un atributo *discriminador*. En la práctica, este problema lleva a la inclusión de valores nulos en la base de datos.

4.3. Iteraciones

Como se menciona en la Sección 3.1 se ha usado la metodología Scrum con *sprints* de dos semanas. Ante lo cual, inicialmente se planificaron ocho iteraciones sin embargo, en base a modificaciones en la planificación propias del desarrollo ágil se han ejecutado un total de 11 iteraciones.

Los siguientes apartados enumeran y describen las iteraciones realizadas.

4.3.1. Iteración 1 - 13 al 26 de mayo

Al ser la primera iteración, es usada para la configuración inicial de elementos como:

- Configuraciones iniciales (repositorio, template proyecto, la integración contínua y entrega contínua -CI/CD por sus siglas en inglés-, contenedores Docker, entre otros).
- Levantamiento de Historias de Usuario.
- Diseño inicial del modelo de datos.

[2]LinQ: Componente de .Net que agrega capacidades de consulta de forma nativa al lenguaje. Para más información: `https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries`

- Primeras configuraciones de API Rest como *back-end* en Loopback 4[3].

Durante esta iteración no se presenta avances en el prototipo.

4.3.2. Iteración 2 - 27 de mayo al 9 de junio

En esta iteración, se comienza a trabajar en las primeras etapas de prototipo, cumpliendo así con las primeras Historias de Usuario: `E1-S7`, `E2-S1` y `E2-S2`.

Una de las tareas más relevantes es el cambio (o migración) en el *framework* usado en la aplicación *back-end*, de Loopback 4 a .Net Core 2.2. Esto a causa de algunas falencias encontradas en el framework basado en Typescript. Básicamente a causa de que la cuarta versión de Loopback (que desde ahí en adelante funcionaría con Typescript) había sido modificada en gran medida desde su tercera versión (basada en Javascript) y aún quedaban funcionalidades sin implementar en la nueva versión.

Entre otras cosas, durante este *sprint* se realizan las siguientes tareas.

- Finalización del modelo de datos implementado en .Net Core.
- Creación de los primeros controladores de la API Rest en .Net Core.
- Configuración definitiva de CI/CD e integración con Docker Hub.
- Integración entre Angular como aplicación de cliente y .Net Core como servidor sirviendo archivos estáticos.
- Gestión de edificios, salas y carreras.

Para el prototipo se presentan los avances presentados en las Figuras 4.4, 4.5 y 4.6.

[3]Loopback 4: Framework de desarrollo back-end basado en Typescript usado para la creación de APIs Rest y microservicios.

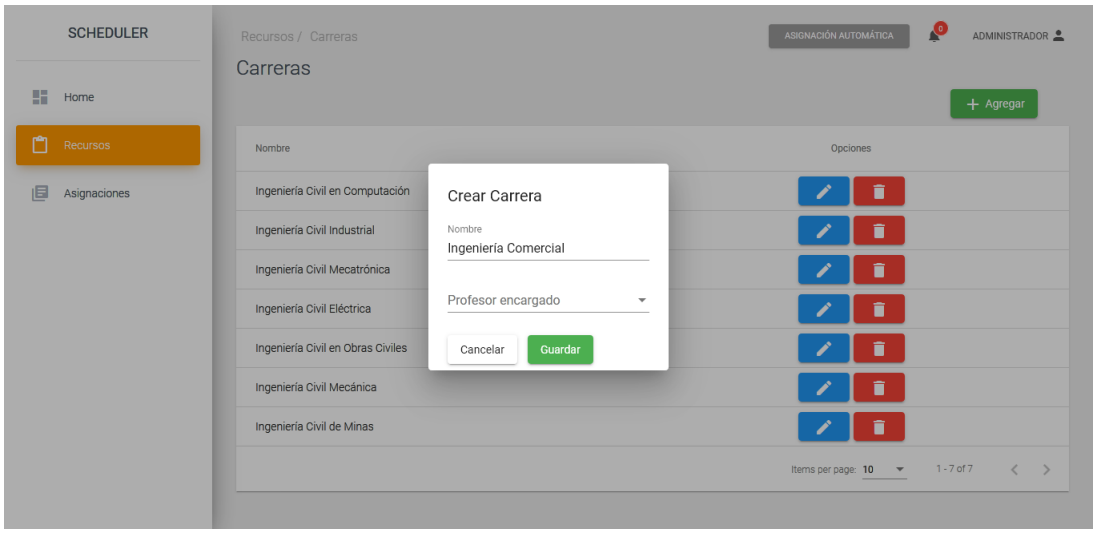


Figura 4.4: Diálogo para crear carreras

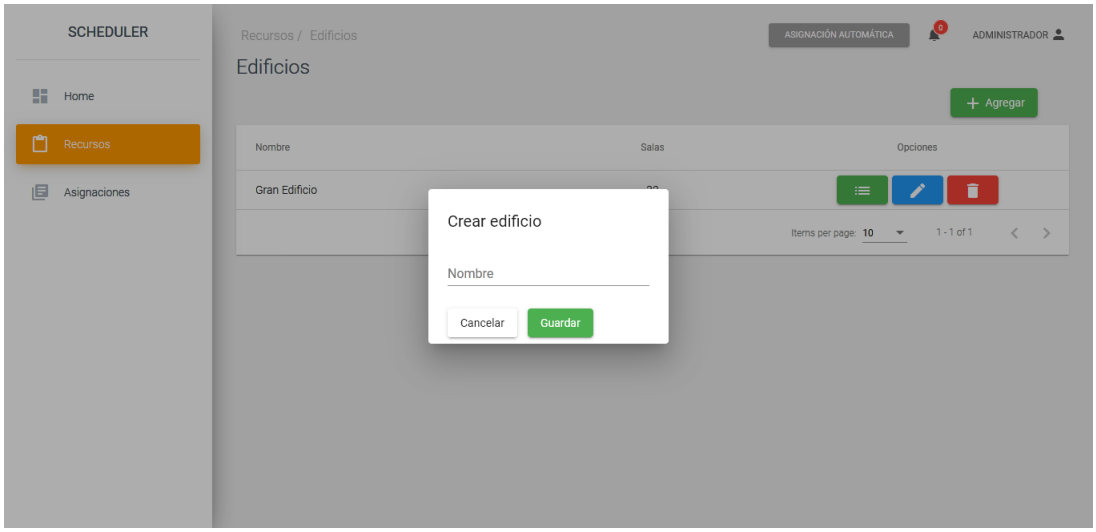


Figura 4.5: Diálogo para crear edificios

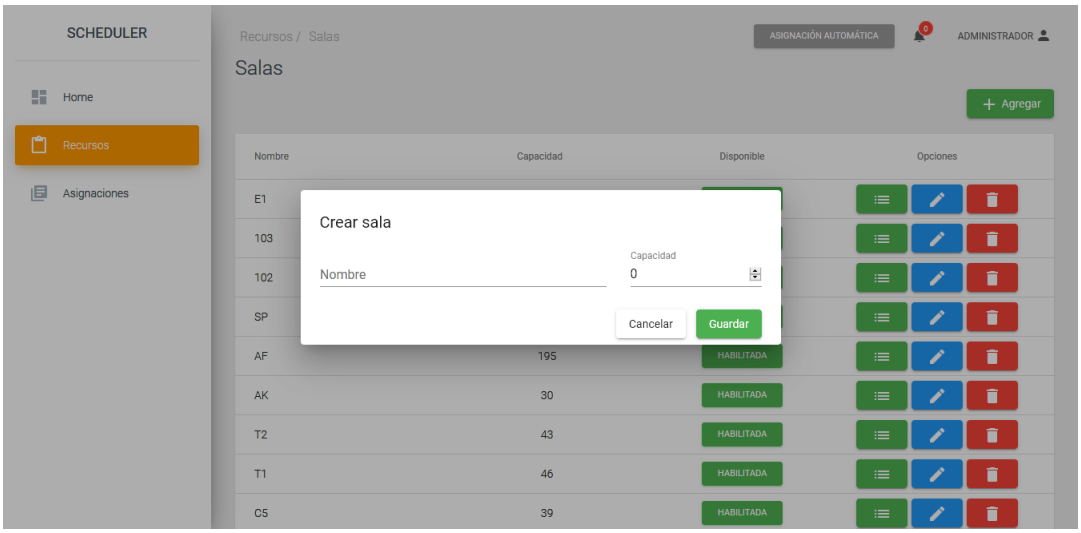


Figura 4.6: Diálogo para crear salas

4.3.3. Iteración 3 - del 10 al 23 de junio

Dado que aún se estaba en las primeras etapas del desarrollo, se decide actualizar Angular a su octava versión con la finalidad de mantener todas las dependencias del proyecto en su versión más actualizada, evitando así posibles problemas y/o vulnerabilidades. Esta actualización no implica mayores problemas debido al instructivo provisto en la plataforma de Angular para realizar una actualización con mayor facilidad.

Además, este *sprint* se usa para probar por primera vez el paso a "producción" en un servidor de pruebas. Este proceso tardó un poco más de lo esperado debido a un problema no previsto entre la conexión desde el contenedor Docker (servidor web) y la instancia de base de datos instalada en el Host del servidor.

Esto último conlleva a que no hubieron avances mayores con respecto a las funcionalidades del sistema.

4.3.4. Iteración 4 - del 24 de junio al 7 de julio

En este *sprint* se completan las Historias de Usuario `E1-S1` y `E1-S2`. Lo que implica que en esta iteración la nueva funcionalidad permite la creación de nuevos módulos y secciones dentro de una carrera, con sus respectivas cantidades de alumnos inscritos o estimados.

Además, por cada sección se permite la selección de su horario. Indicando en qué bloques de qué días dicha sección requerirá de una sala asignada. Iniciando de esta forma los primeros avances para la completitud de la Historia de Usuario `E1-S3`.

Las Figura 4.7 muestra el avance de la interfaz de creación de ramos y secciones en el sistema.

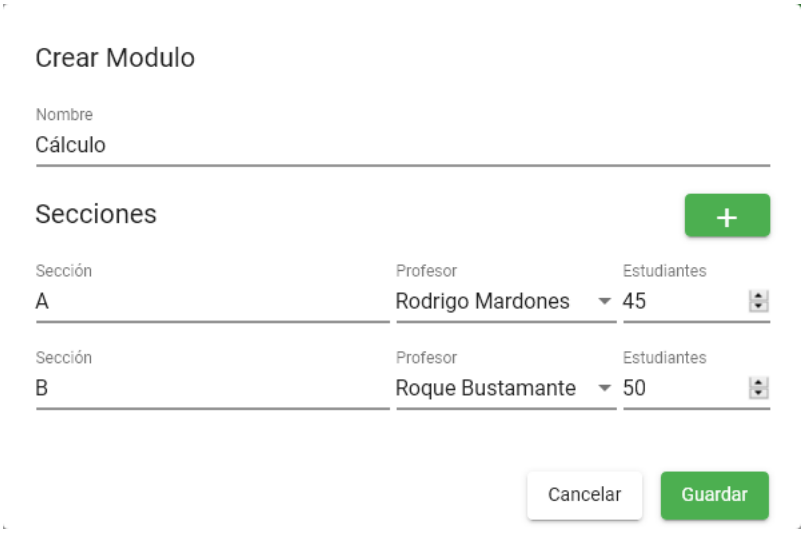


Figura 4.7: Diálogo para crear módulos/ramos con sus secciones

4.3.5. Iteración 5 - del 8 al 21 de julio

Durante esta iteración se cumple con la Historia de Usuario `E1-S6`, además de varias tareas independientes.

Las principales tareas realizadas en este *sprint* son:

- Inicio de sesión usando Identity de .Net Core.
- Registro de nuevos profesores en el sistema.
- Autenticación basada en roles.
- *Seeder* o cargado automático de datos de pruebas.
- Vistas dependiendo del rol del usuario.

- Señalar el proceso de asignación de horario de una carrera como completado (como se muestra en la Figura 4.9).

La vista de inicio de sesión al sistema se muestra en la Figura 4.8.

Figura 4.8: Vista de inicio de sesión

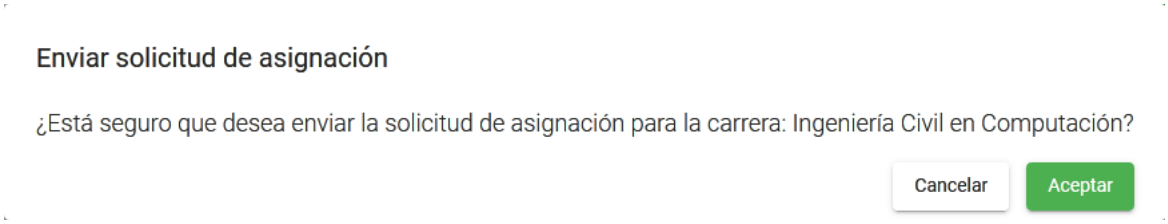


Figura 4.9: Diálogo para enviar solicitud de asignación de salas a ramos

4.3.6. Iteración 6 - del 22 de julio al 4 de agosto

Para esta iteración, se completa definitivamente la Historia de Usuario `E1-S3` agregando mejoras visuales y funcionales con respecto a lo realizado en la Iteración 4 (la vista del prototipo se muestra en la Figura 4.10).

Además, en esta etapa del desarrollo, se descarta las Historia `E1-S5` con la finalidad de acotar el contexto a salas de clases y no laboratorios. La Historia de Usuario `E3-S5` se da como completada en esta iteración, completando implícitamente la Historia `E1-S4`.

4.3.7. Iteración 7 - del 5 al 18 de agosto

La séptima iteración se centra completamente en la implementación del algoritmo para la asignación automática de salas (Historia de Usuario `E3-S7`). La

implementación de dicho algoritmo se revisa más adelante en este mismo capítulo.

Dentro de la interfaz del sistema se agrega un diálogo que permite elegir qué tipo de asignación se desea: Eliminar las asignaciones actuales y comenzar desde cero o Asignar las secciones restantes (ver Figura 4.11).

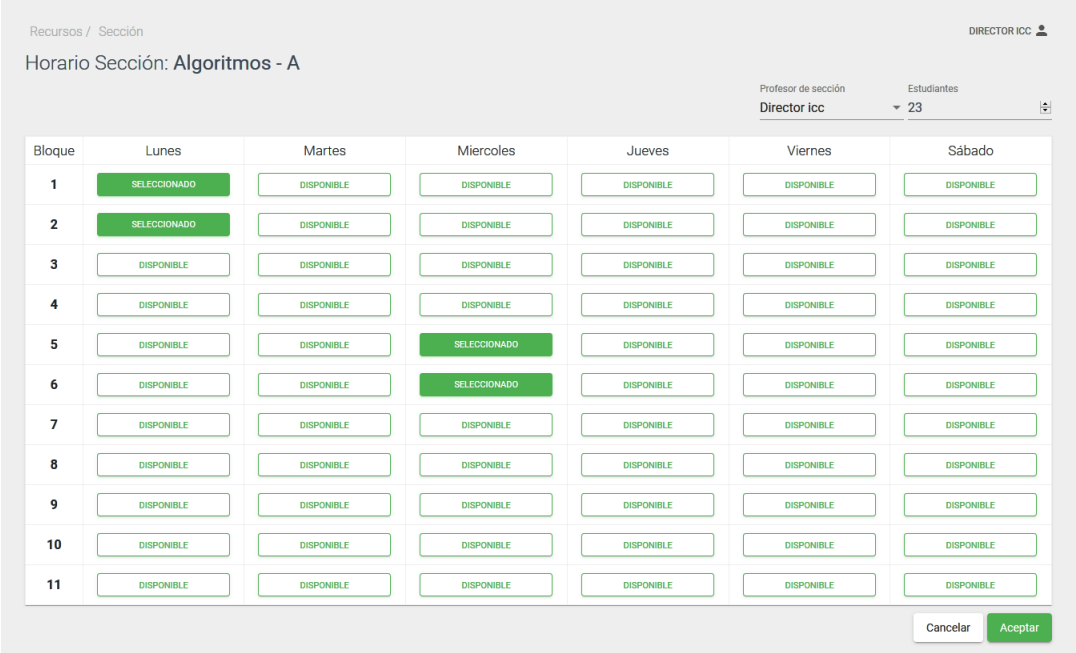


Figura 4.10: Vista de asignación de horario a sección

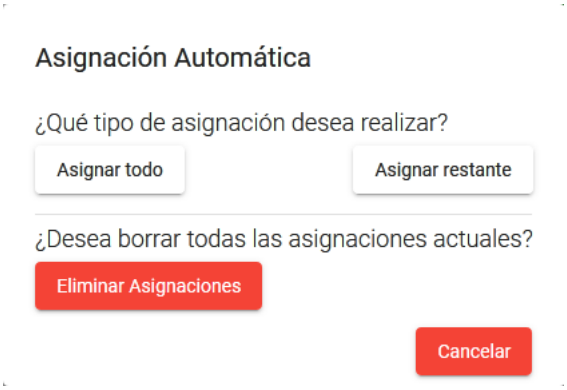


Figura 4.11: Diálogo de asignación semi-automática

4.3.8. Iteración 8 - del 19 de agosto al 1 de septiembre

El desarrollo durante esta iteración se basa en permitir al usuario realizar una asignación manual de salas a bloques de horario en base a sugerencias de salas disponibles. Producto de la ejecución de este *sprint* fue la completitud de la Historia de Usuario `E3-S8`.

La Figura 4.12 muestra el diálogo creado que permite asignar sala manualmente a un bloque o grupo de bloques.

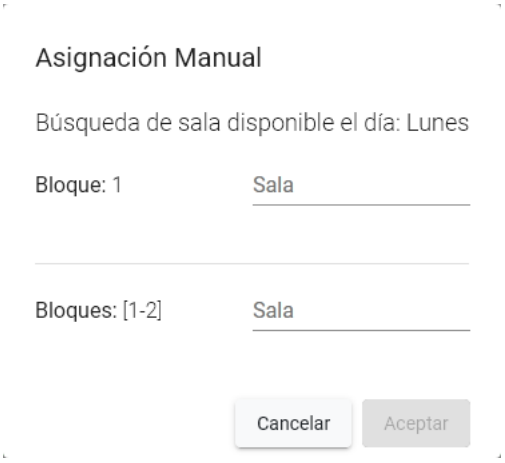


Figura 4.12: Diálogo para la asignación manual de sala a un ramo

4.3.9. Iteración 9 - del 2 al 16 de septiembre

Durante esta iteración se trabaja en completar las Historias de Usuario `E3-S2` y `E3-S3` que implican la creación de solicitudes de asignación *Extraordinaria* y *Especial*.

Además, para acotar los tiempos de desarrollo, se decidió descartar la Historia `E3-S4`.

Como avance del prototipo se presenta (entre otros) el diálogo en la Figura 4.13 donde se permite indicar el día, bloque y duración del grupo de bloques de una solicitud extraordinaria. Además, en la Figura 4.14 se muestra el diálogo que permite realizar una solicitud de asignación especial, indicando el día y bloque del evento especial.

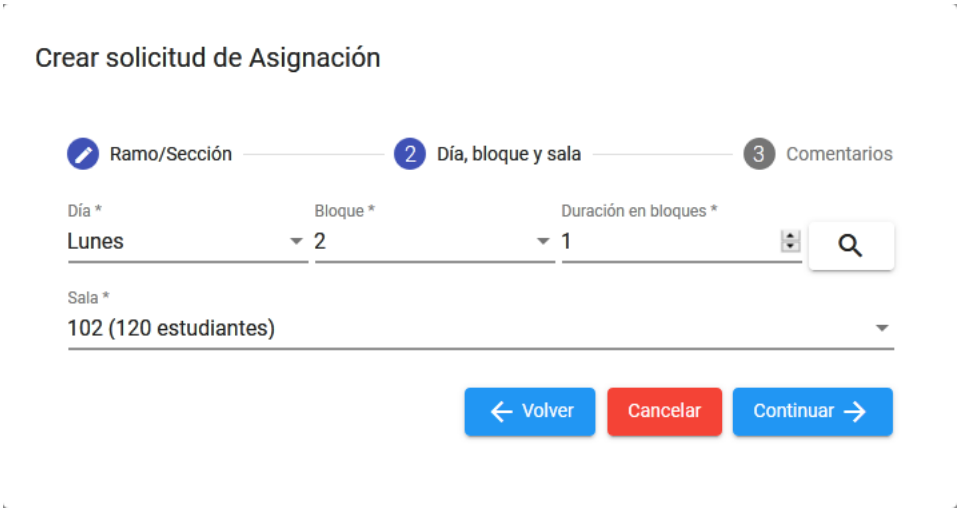


Figura 4.13: Diálogo de solicitud extraordinaria

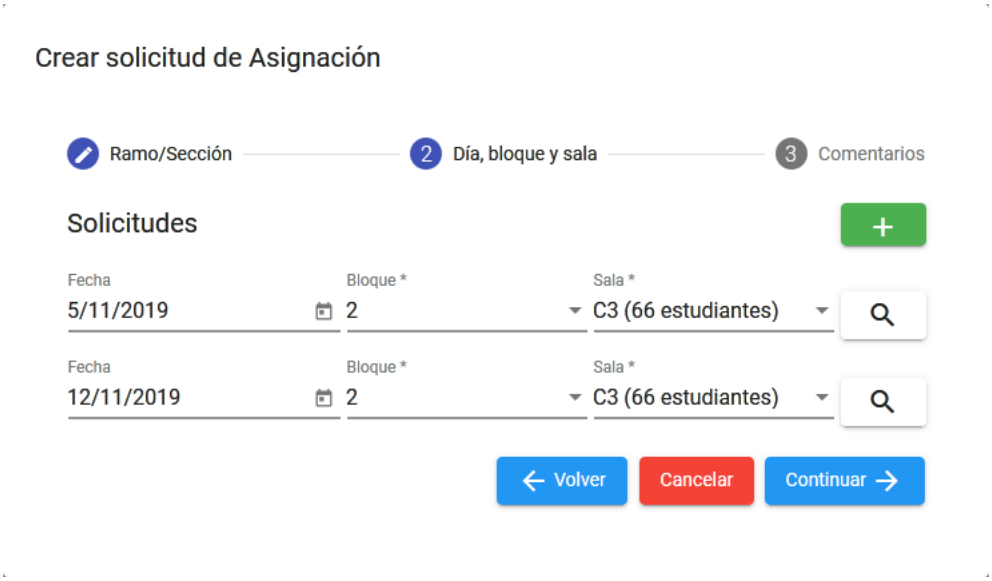


Figura 4.14: Diálogo de solicitud especial

4.3.10. Iteración 10 - del 17 al 30 de septiembre

Durante la décima iteración se completan las Historias `E3-S9` y `E3-S10`. Agregando así, las funcionalidades de aceptar o rechazar las solicitudes del tipo *Extraordinario* y *Especial*.

En este *sprint* se crea un el módulo para la gestión de solicitudes, el cual se puede apreciar en la Figura 4.15.

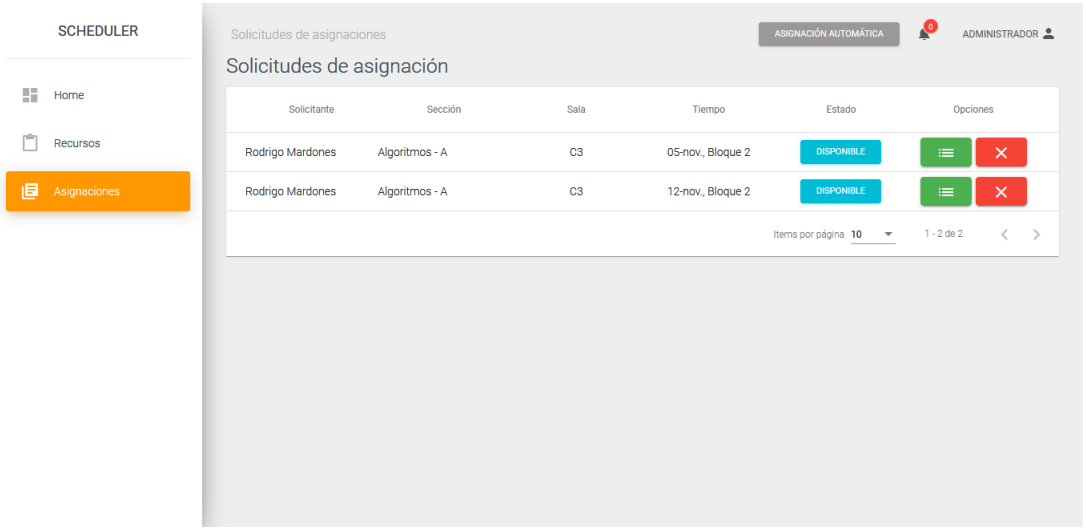


Figura 4.15: Vista de módulo de gestión de solicitudes

4.3.11. Iteración 11 - del 1 al 8 de octubre

Durante la última iteración se completan las Historias `E3-S1` y `E3-S5`. Las funcionalidades completadas en este *sprint* agregan la posibilidad de acceder al sistema sin la necesidad de iniciar sesión como profesor o administrador y revisar si existen salas vacías.

4.4. Algoritmo para la asignación de salas

El método para la asignación de salas se basa en los algoritmos *DoTheMath* y *SearchAndSave*.

Primero, el Algoritmo 1 (denominado *DoTheMath*) es el encargado de buscar en todas las secciones los grupos de bloques[4] que necesitan de la asignación de una sala. Luego, usa el Algoritmo 2, el cual realiza el resto del proceso de asignación. Cabe mencionar que el nombre de este algoritmo no es representativo de su funcionalidad.

El Algoritmo 2 (llamado *SearchAndSave*) es el encargado de buscar una sala que tenga disponibilidad suficiente para cubrir todo un grupo de bloques. Esto último es de suma importancia, ya que se omiten todas las salas que no tienen la disponibilidad necesaria para cubrir el grupo en su totalidad.

[4]Se denomina un grupo de bloques a la serie de bloques consecutivos de una misma sección.

Es importante mencionar que la aplicación conjunta de ambos algoritmos tiene como salida una asignación "semi-automática" de las salas en tan solo un par de minutos. Se trata de una asignación "semi-automática" o "parcial" ya que al terminar su ejecución aún pueden quedar módulos/secciones por asignar. Esto se debe a que realiza asignaciones solamente en base a los grupos de bloques, sin separar dichos bloques en diferentes salas, principalmente debido a que una decisión de este tipo debe ser tomada según la experiencia del encargado y el consentimiento del profesor del ramo.

También cabe recalcar que las asignaciones realizadas por este método puede ser modificadas manualmente (además de realizar nuevas asignaciones manuales) una vez se haya completado el proceso.

Algoritmo 1: DoTheMath

```
1  DoTheMath (Database db)start
2      sections ← db.Sections;
3      classrooms ← db.Classrooms.Where(a => a.Available);
4      sections ← sections.Sort() ;    // by number of students, descending
5      classrroms ← classrroms.Sort() ;          // by capacity, ascending
6      foreach section in sections do
7          for day ← Monday to Saturday do
8              for block ← 0 to 10 do
9                  if section needs a classroom in day and block then
10                     if classrrom not assigned yet in day and block then
                           // Count the blocks (or span)
11                         count ← 0;
12                         endBlock ← block;
13                         while still needing a classroom on (block + endBlock) do
14                             count ← count + 1;
15                             endBlock ← endBlock + 1;
16                         SearchAndSave(db, classrooms, section, day, block, count);
                           // Skip blocks already considered
17                         block ← block + count;
```

Algoritmo 2: SearchAndSave

```
1  SearchAndSave (db, classrooms, section, day, block, span)start
2      index ← |classrooms| − 1;
3      repeat
4          classroom ← classrooms[index];
5          if classroom has not enough space then break ;
6          if not classroom already in use then
7              i ← span; currentBlock ← block; available ← true;
8              while i > 0 do more blocks are needed
9                  if classroom is not in use (day, block) then classroom is available
10                     i ← i − 1;                    // one block less to check
11                     currentBlock ← currentBlock + 1;     // check next block
12                 else classroom is being used
13                     index ← index − 1;              // try on next classroom
14                     available ← false;          // give up on this classroom
15                     break;
16             if available then
17                 for j ← 0 to span do
18                     classroom.MarkBlock(day, block + j, true);
19                     assignation ← newAssignation(...);
20                     db.Assignations.append(assignation);
21                 SaveChanges();
22                 return true;
23         else index ← index − 1;
24     until index ≥ 0;
25     return false;
```

5. Pruebas y resultados

En la Sección 2.4, se menciona el uso de 2 tipos de validaciones: pruebas de caja negra y pruebas de usabilidad (SUS). En este capítulo, se aborda la definición, aplicación y análisis de las mismas.

5.1. Pruebas de caja negra

En esta sección se describen formalmente las pruebas de caja negra a realizar. Sin embargo, los antecedentes de las pruebas realizadas están en el apartado de anexos del documento.

Cabe destacar que las pruebas de caja negra están orientadas a evidenciar el cumplimiento de las historias de usuario.

Por temas de practicidad, en esta sección se entrega la definición formal sólo de una de las pruebas de caja negra. Para el resto de pruebas sólo se entrega la descripción en este capítulo y su definición completa se entrega en el Apéndice A.

Tabla 5.1: Prueba #1 caja negra - Registro de carrera

Código:	T1	
Historia/s de usuario:	E1-S7	
Descripción:	Usuario administrador (encargado/a) puede agregar una nueva carrera en el sistema.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Nombre de carrera. ▪ Nombre de usuario director.	▪ Carrera registrada en la base de datos. ▪ Actualización de lista de carreras.	▪ Carrera guardada exitosamente. ▪ La lista de carreras se actualiza.

La descripción del resto de pruebas de caja negra es la siguiente:

- **T2**: Usuario administrador puede registrar una nueva sala en el sistema.
- **T3**: Usuario administrador (encargado/a) o director de carrera registra un nuevo módulo en el sistema con sus respectivas secciones.
- **T4**: Usuario director de carrera asigna un horario a una sección específica.
- **T5**: Usuario director de carrera indica que ya completó el registro de módulos y horarios, denotando así que es necesaria la asignación de salas.
- **T6**: Usuario estudiante o externo tiene acceso al sistema sin credenciales de acceso.
- **T7**: Usuario profesor envía una solicitud extraordinaria al administrador.
- **T8**: Usuario profesor envía una solicitud especial al administrador.
- **T9**: Usuario administrador realiza la asignación semi-automática de salas a módulos en el sistema.

- **T10**: Usuario administrador realiza la asignación manual de sala a un módulo en el sistema.
- **T11**: Usuario administrador acepta solicitudes de asignación hechas por profesores.
- **T12**: Usuario administrador rechaza solicitudes de asignación hechas por profesores.

5.2. Pruebas de usabilidad SUS

La prueba de usabilidad aplicada en este proyecto está basada en el *framework* SUS (System Usability Scale). SUS plantea un formato de diez preguntas evaluadas en una escala de Likert con valores de uno ("Totalmente desacuerdo") a cinco ("Totalmente de acuerdo") [23]. En esta ocasión, el conjunto de preguntas o ítems a evaluar se ha modificado levemente, sin cambiar el sentido de las preguntas. De esta forma la encuesta está más orientada al contexto específico del proyecto.

A continuación, las Figuras 5.1 y 5.2 definen el formato de la prueba de usabilidad usada.

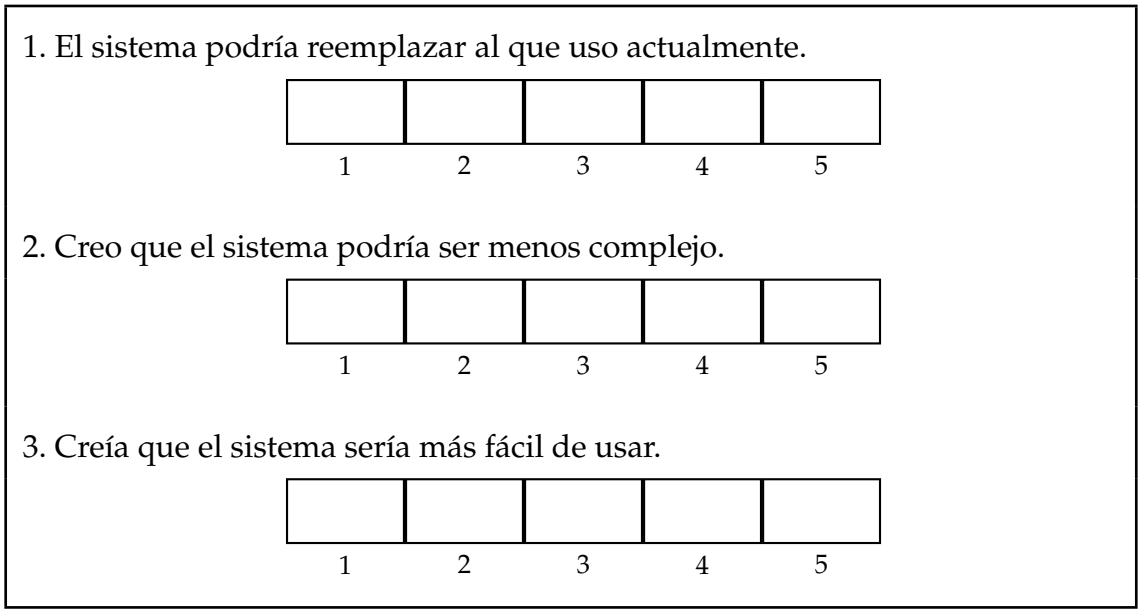


Figura 5.1: Formato prueba de usabilidad utilizando SUS modificada (pt. 1)

4. Necesito la ayuda técnica (de un experto) para usar el sistema.

1	2	3	4	5

5. Las distintas funciones del sistema se complementan entre ellas.

1	2	3	4	5

6. Todas las cosas parecidas funcionan de la misma manera (no hay inconsistencias).

1	2	3	4	5

7. Creo que el sistema es lo suficientemente simple como para que cualquiera lo use rápidamente.

1	2	3	4	5

8. El sistema hace algunas cosas de manera muy engorrosa.

1	2	3	4	5

9. Puedo usar las funcionalidades del sistema sin temor a equivocarme.

1	2	3	4	5

10. Tengo que estudiar aspectos técnicos para poder usar bien el sistema.

1	2	3	4	5

Figura 5.2: Formato prueba de usabilidad utilizando SUS modificada (pt. 2)

5.3. Resultados pruebas de caja negra

Esta sección demuestra la ejecución de las pruebas de caja negra y la completitud de la mismas. Para minimizar la extensión de este documento en este apartado sólo se incluye la primera de ellas, por otro lado, en el Anexo B se incluyen las pruebas restantes que demuestran la aprobación exitosa de la totalidad de pruebas de caja negra descritas en la Sección 5.1.

5.3.1. Prueba T1

La Figura 5.3 muestra los datos de entrada para la prueba descrita en la Tabla 5.1 (nombre de carrera: Ingeniería Civil en Computación y el nombre del profesor encargado: "Director icc"). Por su parte la Figura 5.4 muestra la salida obtenida, el registro correcto de la carrera en la tabla de carreras.

Figura 5.3: T1 - Datos de entrada

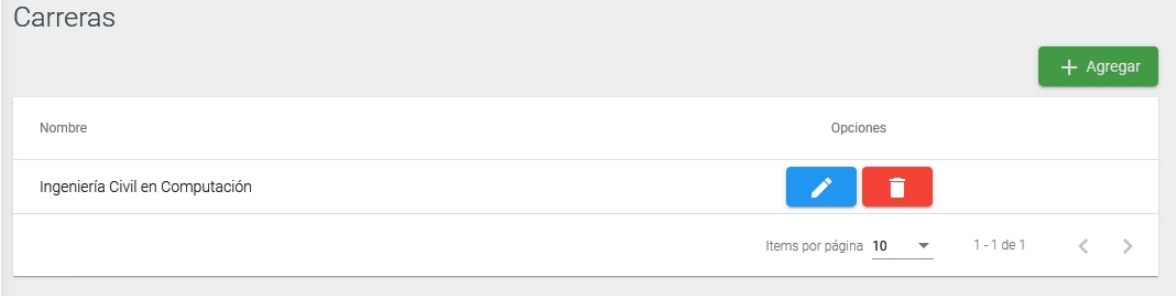


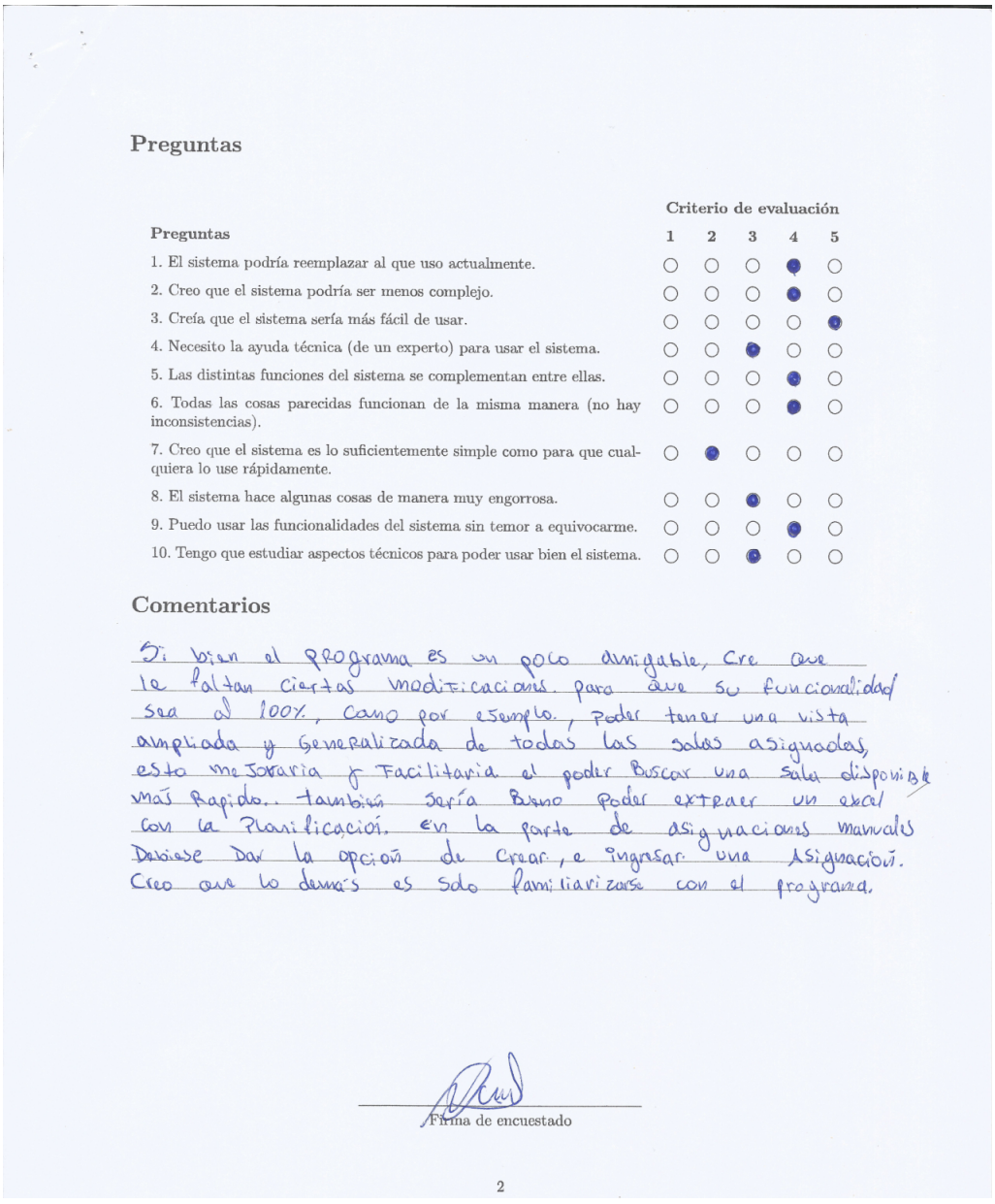
Figura 5.4: T1 - Salida obtenida

En la especificación de cada prueba de caja negra, se indica a qué historia(s) de usuario corresponde. Esto se diseñó de tal forma que cada prueba aprobada implica la completitud de las funcionalidades cubiertas por dicha historia de usuario.

5.4. Resultados prueba de usabilidad SUS

Se decidió aplicar la encuesta de usabilidad a la actual encargada (Ninett Vergara) de la gestión y asignación de salas del Campus Curicó. Evidentemente, eso resta valor estadístico a la validación del sistema, pero se contrarresta con el hecho de que se trata de usuario principal del sistema. Esto se debe a que la plataforma está diseñada para que pudiese ser utilizada únicamente por el encargado/a (en caso de no estar disponible para los demás usuarios).

La figura 5.5 corresponde a la segunda parte de la encuesta aplicada, que muestra las respuestas obtenidas y los comentarios obtenidos. En el Anexo C se incluye la primera parte de la encuesta.



Preguntas

Preguntas	Criterio de evaluación 1	2	3	4	5
1. El sistema podría reemplazar al que uso actualmente.	○	○	○	●	○
2. Creo que el sistema podría ser menos complejo.	○	○	○	●	○
3. Creía que el sistema sería más fácil de usar.	○	○	○	○	●
4. Necesito la ayuda técnica (de un experto) para usar el sistema.	○	○	●	○	○
5. Las distintas funciones del sistema se complementan entre ellas.	○	○	○	●	○
6. Todas las cosas parecidas funcionan de la misma manera (no hay inconsistencias).	○	○	○	●	○
7. Creo que el sistema es lo suficientemente simple como para que cualquiera lo use rápidamente.	○	●	○	○	○
8. El sistema hace algunas cosas de manera muy engorrosa.	○	○	●	○	○
9. Puedo usar las funcionalidades del sistema sin temor a equivocarme.	○	○	○	●	○
10. Tengo que estudiar aspectos técnicos para poder usar bien el sistema.	○	○	●	○	○

Comentarios

Si bien el programa es un poco amigable, Cree que le faltan ciertas modificaciones para que su funcionalidad sea al 100%, como por ejemplo, poder tener una vista ampliada y Generalizada de todas las salas asignadas, esto mejoraria y Facilitaria el poder Buscar una sala disponible más Rapido.. también sería Bueno poder extraer un excel con la Planificación. en la parte de asignaciones manuales Deviase Dar la opción de Crear, e ingresar una Asignación. Creo que lo demás es Solo familiarizarse con el programa.

Firma de encuestado

2

Figura 5.5: Prueba de usabilidad - Hoja 2, firmada por Ninett Vergara

El resultado obtenido (de un total de 100 puntos) es el siguiente:

$$Puntaje = 22 * 2,5 = 55$$

Los resultados de la prueba indican que la usabilidad del sistema *"está correcto, pero queda bastante por mejorar"*. La encuestada, en la sección de comentarios, hace

referencia a 3 elementos o funcionalidades que a su preferencia faltan para que el sistema esté completo y pueda aumentar su apreciación sobre el mismo (*"... para que su funcionalidad sea al 100%"*).

1. Tener una vista general de las salas y no sólo una individual de cada una de ellas. Lo cual facilita la búsqueda de salas disponibles independientemente de una módulo.

2. La entrega de un reporte no sólo de los módulos con asignaciones, sino que también con la planificación de las salas.

3. La facilidad de poder crear una asignación manual (especial o extraordinaria) sin la necesidad de que un profesor envíe una solicitud.

Además, la encuestada destaca la importancia de familiarizarse con el nuevo sistema de asignación de salas y evitar el posible *"rechazo a las nuevas tecnologías"* debido al largo tiempo que lleva usando su planilla actual.

Posteriormente, las solicitudes fueron agregadas como nuevas funcionalidades del sistema en sí. Lamentablemente, no fue posible acordar una nueva reunión para volver a analizar el sistema y ver el impacto en los resultados al aplicar una nueva encuesta teniendo en consideración las nuevas funcionalidades.

6. Conclusiones y trabajo futuro

Como se describió en el primer apartado, en El Campus Curicó de la Universidad de Talca tenemos presente un problema del tipo "CPR" (o problema de asignación de recursos comúnes). Este se centra en la asignación semestral de salas para la realización de cátedras, talleres, ayudantías o laboratorios, entre otros. Al día de hoy, este problema se resuelve usando una planilla Excel para el registro de salas, bloques y asignaturas. Al tratarse de una cantidad considerable (dentro de los parámetros que un humano puede manejar facilmente) de registros, el uso de Excel hace que esta solución sea poco eficiente.

En base a lo anterior, es necesario implementar un nuevo sistema que pueda suplir las necesidades tanto del Campus Curicó como del encargado/a de realizar las asignaciones semestralmente.

La presente memoria detalla el diseño, metodología y desarrollo de una plataforma web para la gestión de salas dentro del Campus Curicó. Esta plataforma busca solventar las carencias de la metodología actual de trabajo, así como ayudar en la eficiencia y facilitación del proceso. En aspectos generales, el sistema permite la administración de salas y módulos/secciones de carreras y/o departamentos. También permite la asignación del recurso "sala" durante un tiempo determinado "bloque", a un evento "sección" en específico . Además, permite la gestión de solicitudes de asignación basadas en eventos especiales por tiempo acotado o por toda la duración del semestre.

El sistema en cuestión se ha validado usando pruebas de caja negra, las cuales han sido definidas de forma tal que cada una de ellas está directamente relacionada con al menos una historia de usuario. Por ende, la aprobación de todas las pruebas de caja negra implica que la plataforma cumple con la totalidad de las

funcionalidades solicitadas/definidas como historias de usuario. Específicamente, el 100% de las pruebas de caja negra fueron aprobadas, es decir, el sistema efectivamente cumple con todas las funcionalidades requeridas.

Adicionalmente, cabe destacar la notable mejora referente al tiempo que se puede tardar un encargado en realizar una asignación completa. Usando la plataforma anterior, el proceso completo puede tardar desde dos semanas hasta un mes de trabajo. Por otro lado, usando la nueva plataforma, realizando una asignación semi-automática se logra una gran parte de las asignaciones en tan solo minutos. Cumpliendo de esta forma el objetivo principal de esta memoria.

Además, se ha aplicado una encuesta de usabilidad SUS con una puntuación de 55pts. de un máximo de 100pts., implicando que *"está correcto, pero queda bastante por mejorar"*. Los resultados obtenidos en esta encuesta fueron de vital importancia para identificar funcionalidades críticas solicitadas por el usuario principal que no fueron detectadas inicialmente. Dichas funcionalidades faltantes fueron agregadas posteriormente al producto final, por lo que frente a una futura nueva aplicación de este método de evaluación, arrojaría mejores resultados.

A modo de desglose del cumplimiento de los objetivos específicos propuestos se tiene lo siguiente:

1. Objetivo *"Facilitar la asignación de salas a módulos..."* se cumple en su totalidad, ya que se entrega un sistema capaz de asignar automáticamente a gran parte de las secciones de un semestre común en un periodo de tiempo mucho menor al que tarda todo el proceso original.

2. Objetivo *"Facilitar la consulta y asignación de salas para actividades extraordinarias..."* se cumple en su totalidad, ya que se presenta un módulo para realizar y gestionar las solicitudes de asignación, tanto para eventos durante todo un semestre como eventos de duración variable. Además, se proveen métodos para la consulta de salas disponibles y la posibilidad de exportar esta información en formato Excel.

3. Objetivo *"Facilitar gestión de solicitudes de cambios y/o asignación de salas"* se cumple en su totalidad, ya que la forma provee un método para la asignación manual de salas, indicando como posibles alternativas sólo aquellas salas disponibles durante los bloques requeridos.

Lo anterior está respaldado con los comentarios entregados por el usuario principal del sistema y las posteriores adiciones en cuanto a funcionalidades que se implementaron.

Por lo tanto, basándose en los resultados de las pruebas se puede afirmar que el sistema cumple con los requerimientos o historias de usuario definidas (aunque podría mejorar en términos de usabilidad) además de los objetivos específicos propuestos. Por consiguiente, también cumple a cabalidad con el objetivo general de este proyecto.

Finalmente, a modo de trabajo futuro se propone la integración con los demás sistemas de la universidad como el registro de módulos a dictar durante el semestre o el registro de profesores del campus. Adicionalmente, se propone la generalización del sistema para que este pueda ser utilizado en otros contextos u otros campus de otras universidades del país. Además, se propone la implementación del soporte para casos especiales en los que (por ejemplo) una sección de un módulo se divide en sub-secciones para realizar los diferentes laboratorios del módulo.

Bibliografía

[1] E. Ostrom. *Governing the Commons: The Evolution of Institutions for Collective Action*. Ed. por Cambridge University Press. 1990.

[2] R. Margaret. *What is RESTful API? - Definition from WhatIs.com*. URL: https://searchmicroservices.techtarget.com/definition/RESTful-API. (accedido: 23-06-2019).

[3] Collins Dictionary. *Framework definition and meaning?* URL: https://www.collinsdictionary.com/dictionary/english/framework. (accedido: 18-06-2019).

[4] M. Steven. *Single Page Application y su funcionamiento*. URL: https://spiradreams.com/como-funciona-una-single-page-application/. (accedido: 18-06-2019).

[5] Techopedia. *What is Object-Relational Mapping (ORM)?* URL: https://www.techopedia.com/definition/24200/object-relational-mapping--orm. (accedido: 18-06-2019).

[6] Clustrix. *ACID Compliance: What It Means and Why You Should Care*. URL: https://www.clustrix.com/bettersql/acid-compliance-means-care/. (accedido: 22-06-2019).

[7] C. Rodway. *Classroom Bookings*. Gateshead, UK, 2006. URL: https://www.classroombookings.com/. (accedido: 05-04-2020).

[8] School Cloud Systems. *Room Booking System*. School Cloud Systems, Glasgow G40 3AP, UK, 2019. URL: https://www.roombookingsystem.co.uk. (accedido: 05-04-2020).

[9] Visual Scheduling Systems. *Visual Classroom Scheduler*. New South Whales, 1994. URL: http://www.vss.com.au/index.asp. (accedido: 05-04-2020).

[10] S. Navuduri. *Design and Implementation of a Classroom Allocation System Prototype*. Ed. por Texas A&M University-Corpus Christi The Department of Computing Sciences. 2016.

[11] JReport. *3-Tier Architecture: A Complete Overview*. URL: https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/. (accedido: 23-06-2019).

[12] F. Yakov. *Angular 2 and TypeScript - A High Level Overview*. URL: https://www.infoq.com/articles/Angular2-TypeScript-High-Level-Overview/. (accedido: 23-06-2019).

[13] G. Dave. *The History of Angular*. URL: https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7. (accedido: 23-06-2019).

[14] H. Máté. *AngularJS to Angular - a brief history with some tips to get started!* URL: https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7. (accedido: 23-06-2019).

[15] Facebook Inc. *React - Una biblioteca Javascript para construir interfaces de usuario*. URL: https://es.reactjs.org. (accedido: 18-06-2019).

[16] Y. Evan. *Vue.js*. URL: https://vuejs.org. (accedido: 18-06-2019).

[17] L. Rich. *About .NET Core*. URL: https://docs.microsoft.com/en-us/dotnet/core/about. (accedido: 23-06-2019).

[18] StrongLoop Inc. *Express: Infraestructura web rápida, minimalista y flexible para Node.js*. URL: https://expressjs.com/es/. (accedido: 22-06-2019).

[19] B. David. *Why I Choose PostgreSQL Over MySQL/MariaDB*. URL: https://insights.dice.com/2015/03/19/why-i-choose-postgresql-over-mysqlmariadb/. (accedido: 22-06-2019).

[20] The PostgreSQL Global Development Group. *PostgreSQL: License*. URL: https://www.postgresql.org/about/licence/. (accedido: 22-06-2019).

[21] H.F. Cervone. «Understanding agile project management methods using Scrum». En: *OCLC Systems & Services: International digital library perspectives* 27.1 (2011), págs. 18-22. DOI: 10.1108/10650751111106528. (accedido: 23-06-2019).

[22] J. Michael y L. Walter. *Scrum Reference Card*. URL: http://scrumreferencecard.com/scrum-reference-card/. (accedido: 23-06-2019).

[23] Usability.gov. *System Usability Scale (SUS)*. URL: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html. (accedido: 11-08-2019).

[24] I.E. Allen y C.A. Seaman. «Likert Scales and Data Analyses». En: *Quality Progress, The official publication of ASQ* (jul. de 2007). Ed. por Quality Progress.

[25] Software Testing Class. *What is Black Box Testing?* URL: http://www.softwaretestingclass.com/what-is-black-box-testing/. (accedido: 30-06-2019).

[26] N. Cerpa y J.M. Verner. «Why Did Your Project Fail?» En: *Commun. ACM* 52.12 (dic. de 2009), págs. 130-134. ISSN: 0001-0782. DOI: 10.1145/1610252.1610286.

[27] A. Hudaib y col. «Requirements Prioritization Techniques Comparison». En: *Modern Applied Science* 12 (ene. de 2018). DOI: 10.5539/mas.v12n2p62.

ANEXOS

A. Pruebas de caja negra

Tabla A.1: Prueba #2 caja negra - Registro de sala

Código:	T2	
Historia/s de usuario:	E2-S1, E2-S2	
Descripción:	Usuario administrador puede registrar una nueva sala en el sistema.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Nombre de edificio. ▪ Nombre de sala. ▪ Número de alumnos que pueden entrar en la sala.	▪ Edificio se registra en la base de datos. ▪ Sala se registra en la base de datos. ▪ Actualización de lista de edificios. ▪ Actualización de lista de salas en edificio correspondiente.	▪ Edificio y Sala se registran en la base de datos. ▪ La lista de edificios se actualiza. ▪ La lista de salas dentro de un edificio se actualiza.

Tabla A.2: Prueba #3 caja negra - Registro de módulos

Código:	T3	
Historia/s de usuario:	E1-S1, E1-S2	
Descripción:	Usuario administrador (encargado/a) o director de carrera registra un nuevo módulo en el sistema con sus respectivas secciones.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Carrera a la que pertenece. ▪ Nombre de módulo. ▪ Cantidad de secciones y nombre (o letra) de las mismas. ▪ Profesor de cada sección. ▪ Cantidad de estudiantes por sección.	▪ Secciones se registran exitosamente en la base de datos. ▪ Lista de secciones se actualiza con todas las secciones agregadas.	▪ Secciones registrados exitosamente. ▪ La lista de secciones se actualizada.

Tabla A.3: Prueba #4 caja negra - Registro de horario de sección

Código:	T4	
Historia/s de usuario:	E1-S3, E1-S4	
Descripción:	Usuario administrador (encargado/a) o director de carrera asigna un horario a una sección específica.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Sección a la que se desea registrar su horarios. ▪ Bloques de horario de sección.	▪ Sección registra exitosamente su horario. ▪ Vista de horario se de sección se actualiza con el horario.	▪ Horario de sección registrado exitosamente. ▪ Vista de horario se actualizado correctamente.

Tabla A.4: Prueba #5 caja negra - Envío de solicitud de asignación de salas

Código:	T5	
Historia/s de usuario:	E1-S6, E3-S6	
Descripción:	Usuario director de carrera indica que ya completó el registro de módulos y horarios, denotando así que es necesaria la asignación de salas.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Carrera cuyo horario ha sido completado.	▪ Carrera se registra exitosamente en la base de datos como completada. ▪ Vista se actualiza indicando que la carrera se ha completado. ▪ Vista del encargado muestra notificación de que una carrera se ha completado.	▪ Carrera registrada como completada exitosamente. ▪ Vista se actualiza e indica que la carrera se ha completado. ▪ Vista del encargado indica correctamente que una carrera se ha completado.

Tabla A.5: Prueba #6 caja negra - Acceso al sistema a usuario externos

Código:	T6	
Historia/s de usuario:	E3-S1, E3-S5	
Descripción:	Usuario estudiante o externo tiene acceso al sistema sin credenciales de acceso.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Usuario sin credenciales de acceso.	▪ Usuario puede acceder exitosamente y tiene acceso a listado de salas con su horario.	▪ Usuario accede y tiene acceso a listado de salas con su horario.

Tabla A.6: Prueba #7 caja negra - Envío solicitud extraordinaria

Código:	T7	
Historia/s de usuario:	E3-S2	
Descripción:	Usuario profesor envía una solicitud extraordinaria al administrador.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Sección. ▪ Día, bloque y duración en bloques que se necesita.	▪ Solicitud extraordinaria se registra en la base de datos exitosamente. ▪ Encargado ve una nueva solicitud extraordinaria en el módulo de solicitudes.	▪ La solicitud ha sido registrada en la base de datos. ▪ El encargado ve en la vista de solicitudes la nueva solicitud extraordinaria.

Tabla A.7: Prueba #8 caja negra - Envío solicitud especial

Código:	T8	
Historia/s de usuario:	E3-S3	
Descripción:	Usuario profesor envía una solicitud especial al administrador.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Sección. ▪ Día en el calendario y bloque que se necesita.	▪ Solicitud especial se registra en la base de datos exitosamente. ▪ Encargado ve una nueva solicitud especial en el módulo de solicitudes.	▪ La solicitud ha sido registrada en la base de datos. ▪ El encargado ve en la vista de solicitudes la nueva solicitud especial.

Tabla A.8: Prueba #9 caja negra - Asignación semi-automática de salas

Código:	T9	
Historia/s de usuario:	E3-S7	
Descripción:	Usuario administrador realiza la asignación semi-automática de salas a módulos en el sistema.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Usuario administrador con sesión iniciada.	▪ Módulos con salas asignadas se registran en el sistema y quedan módulos sin asignar.	▪ Módulos cuentan con sala asignada. Quedan módulos sin asignar.

Tabla A.9: Prueba #10 caja negra - Asignación manual de salas

Código:	T10	
Historia/s de usuario:	E3-S8	
Descripción:	Usuario administrador realiza la asignación manual de sala a un módulo en el sistema.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Carrera. ▪ Sección. ▪ Día y bloque a asignar. ▪ Sala disponible en día y bloque.	▪ Asignación de sala manual se registra exitosamente en la base de datos. ▪ Vista de horario de sección se actualiza y muestra el nombre de la sala asignada en cada bloque.	▪ Asignación manual registrada en la base de datos. ▪ La vista de horario muestra correctamente el nombre de la sala asignada en el bloque.

Tabla A.10: Prueba #11 caja negra - Aceptación de solicitudes de asignación

Código:	T11	
Historia/s de usuario:	E3-S9	
Descripción:	Usuario administrador acepta solicitudes de asignación hechas por profesores.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Solicitud de asignación extraordinaria o especial.	▪ Solicitud es aceptada y registra exitosamente en la base de datos. ▪ Horario de sección se actualiza mostrando el nombre de la sala asignada. ▪ Vista de solicitudes se actualiza y muestra la solicitud en estado de aceptada.	▪ Solicitud pasa a estado de aceptado y se registra en la base de datos correctamente. ▪ La vista de horario muestra correctamente el nombre de la sala asignada en el bloque. ▪ Vista de solicitudes se actualiza correctamente con la solicitud en estado de aceptado.

Tabla A.11: Prueba #12 caja negra - Rechazo de solicitudes de asignación

Código:	T12	
Historia/s de usuario:	E3-S10	
Descripción:	Usuario administrador rechaza solicitudes de asignación hechas por profesores.	
Entrada	**Salida esperada**	**Salida obtenida**
▪ Solicitud de asignación extraordinaria o especial.	▪ Solicitud es rechazada y se elimina exitosamente en la base de datos. ▪ La vista de solicitudes se actualiza y la solicitud no vuelve a aparecer.	▪ Solicitud es eliminada de la base de datos. ▪ La vista de solicitudes se actualiza y la solicitud ha sido removida.

B. Resultados pruebas de caja negra

B.1. Prueba T1

B.2. Prueba T2

Las Figuras B.1 y B.2 muestran los datos de entrada (nombre de sala y cantidad de alumnos) y la salida obtenida (sala nueva registrada en la lista de salas) respectivamente.

Figura B.1: T2 - Datos de entrada

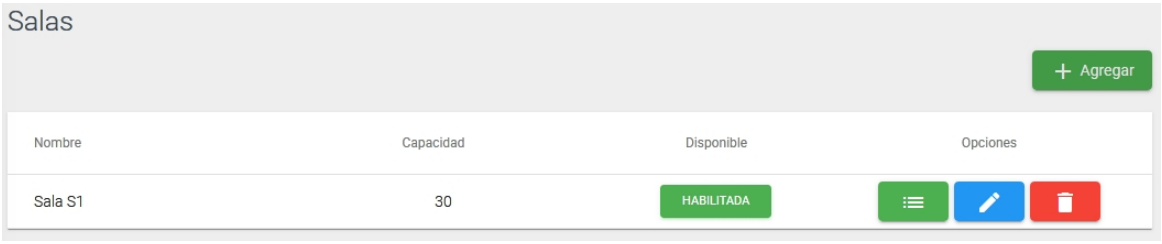


Figura B.2: T2 - Salida obtenida

B.3. Prueba T3

La Figura B.3 muestra la creación según los datos de entrada de dos secciones del módulo "Introducción a la ICC" cada una dictada por un profesor diferente y una cantidad de alumnos distinta. Por otro lado, la Figura B.4 muestra el resultado obtenido, el registro de ambas secciones en el sistema.

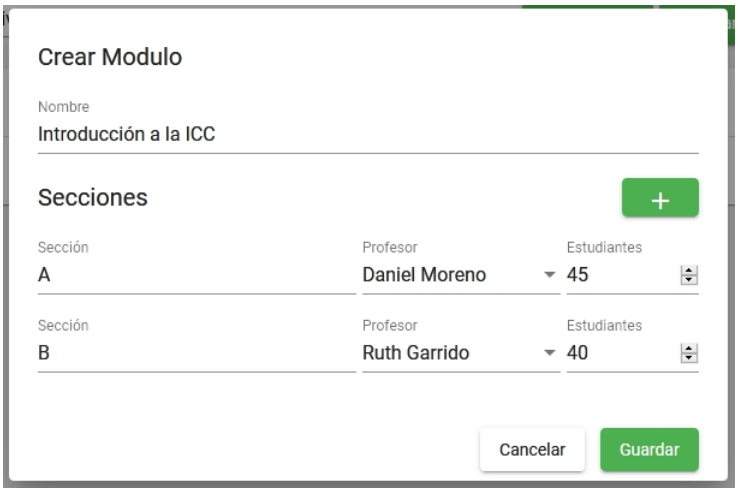


Figura B.3: T3 - Datos de entrada

Figura B.4: T3 - Salida obtenida

B.4. Prueba T4

La Figura B.5 muestra cómo se registra el horario de la sección `A` del módulo "Introducción a la ICC" para los bloques uno y dos del día lunes y los bloques tres y cuatro del día martes.

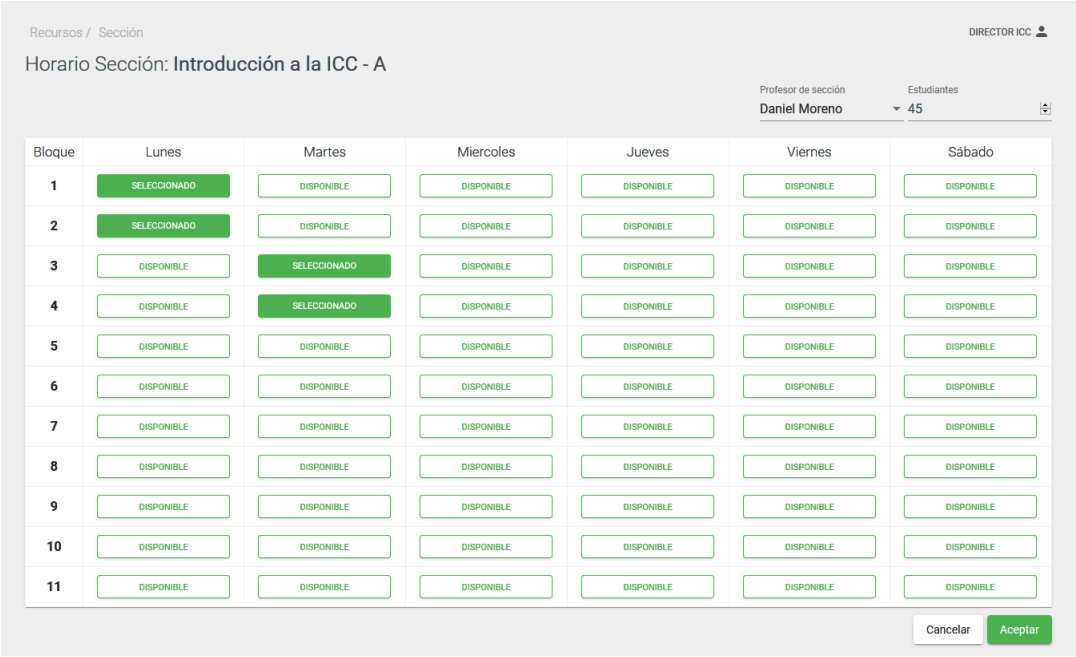


Figura B.5: T4 - Salida obtenida

B.5. Prueba T5

La Figura B.6 muestra el diálogo para la modificación del estado de una carrera dada una entrada (la carrera previamente seleccionada). Por su parte la Figura B.7 muestra el estado actualizado de la carrera indicada.

Figura B.6: T5 - Modificación estado de carrera

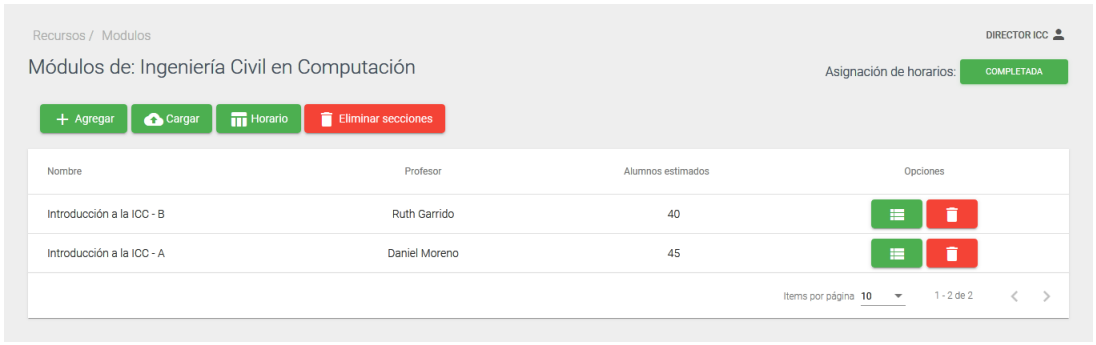


Figura B.7: T5 - Salida obtenida

B.6. Prueba T6

En la Figura B.8 se muestra la vista de inicio de sesión de la plataforma, el cual incluye un apartado para el ingreso al sistema sin credenciales de acceso. La Figura B.9 muestra la opciones habilitadas para un usuario sin credenciales (comúnmente un estudiante). Estas funciones corresponden sólo a poder visualizar la información referente a las salas y su horario así como las secciones y sus horarios con asignaciones.

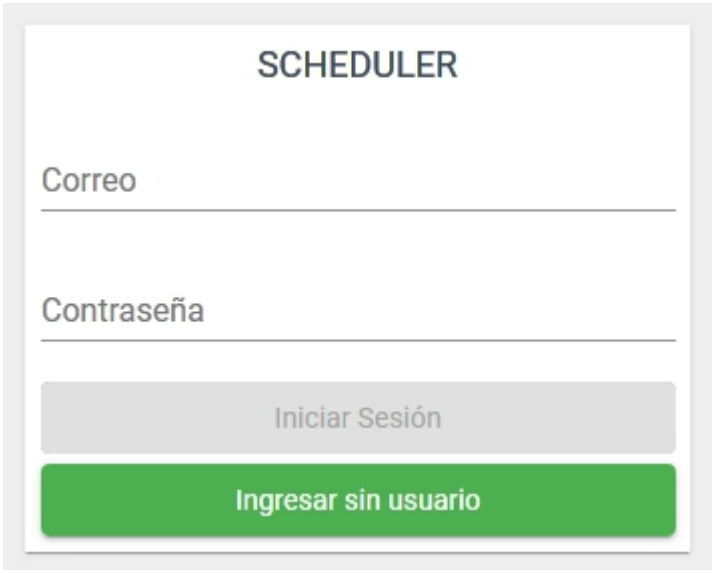


Figura B.8: T6 - Vista de inicio se sesión con opción para ingreso sin credenciales

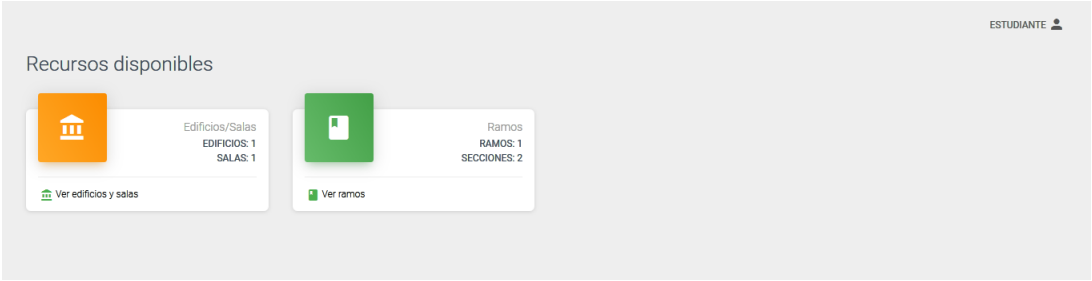


Figura B.9: T6 - Vista de usuario sin credenciales (estudiante) con acceso a opciones limitadas

B.7. Prueba T7

Las Figuras B.10, B.11, B.12 y B.13 muestran los pasos para la creación de una solicitud extraordinaria dados los datos de entrada ejemplificados en las figuras.

Figura B.10: T7 - Selección de tipo de solicitud (extraordinaria)

Figura B.11: T7 - Entrada sección por la que se realiza la solicitud

Figura B.12: T7 - Entrada datos día, bloque y sala de solicitud

Figura B.13: T7 - Entrada comentarios adicionales a solicitud

B.8. Prueba T8

Las Figuras B.14, B.15, B.16 y B.17 muestran los pasos para la creación de una solicitud especial dados los datos de entrada ejemplificados en las figuras.

Por su parte, la Figura B.18 muestra al salida obtenida (el registro de las sol generadas tanto en las pruebas T7 y T8).

Figura B.14: T8 - Selección de tipo de solicitud (especial)

Figura B.15: T8 - Entrada sección por la que se realiza la solicitud

Figura B.16: T8 - Entrada datos día (calendario), bloque y sala de solicitud

Figura B.17: T8 - Entrada comentarios adicionales a solicitud

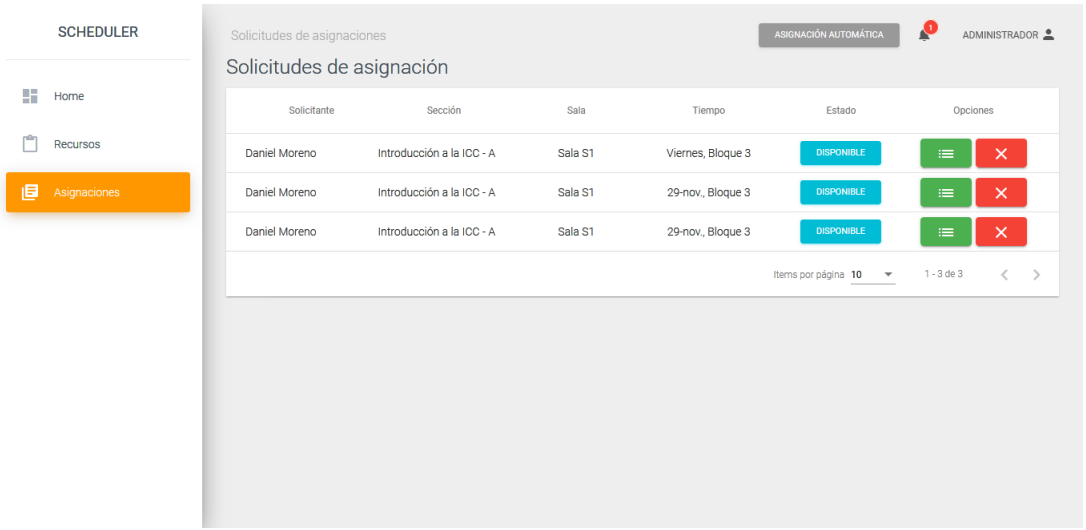


Figura B.18: T7 y T8 - Salida esperada, solicitudes registradas

B.9. Prueba T9

Para el caso acotado de una sala registrada y los módulos de una carrera, se realiza una asignación automática de salas mediante en diálogo de la Figura B.19. Al verificar el horario de la única sala creada se comprueba la salida obtenida (diferentes secciones tienen asignada dicha sala en su horario), como se demuestra en la Figura B.20.

Figura B.19: T9 - Vista de selección de tipo de asignación automática

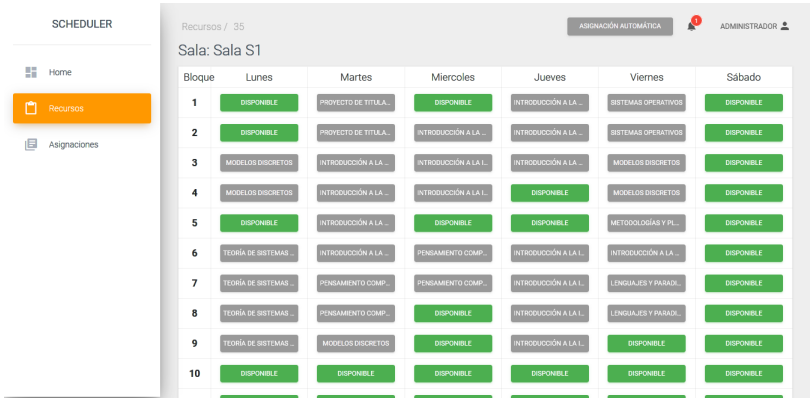


Figura B.20: T9 - Salida esperada, ejemplo de sala cuyos bloques fueron asignados automáticamente

B.10. Prueba T10

En la Figura B.21 se muestra la asignación manual de sala a sección en un determinado grupo de bloques. Por su parte, la Figura B.22 se muestra la salida obtenida (la actualización en la vista de horario con el nombre de la sala asignada).

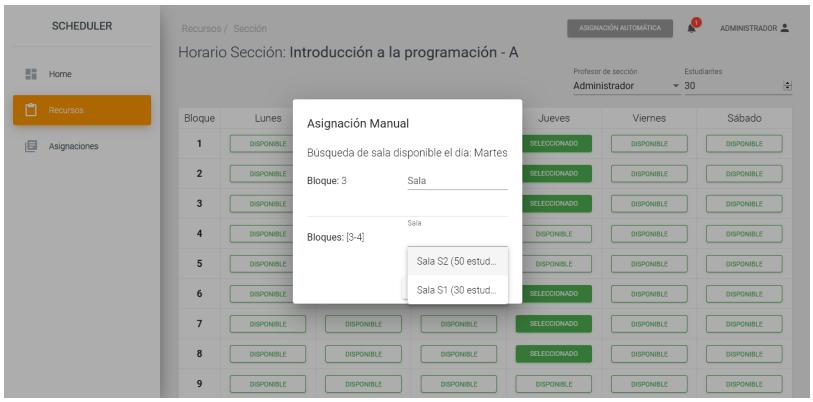


Figura B.21: T10 - Entrada de sección seleccionada y sala elegida

Figura B.22: T10 - Salida esperada, sección registra sala asignada en bloques indicados

B.11. Prueba T11

La Figura B.23 muestra la entrada o vista de selección de una solicitud extraordinaria de asignación que se aceptará. La Figura B.24 muestra la salida obtenida, la actualización de los estados de la solicitud aceptada y de las demás solicitudes que puedan entrar en conflicto.

Figura B.23: T11 - Entrada solicitud especial o extraordinaria seleccionada para ser aceptada

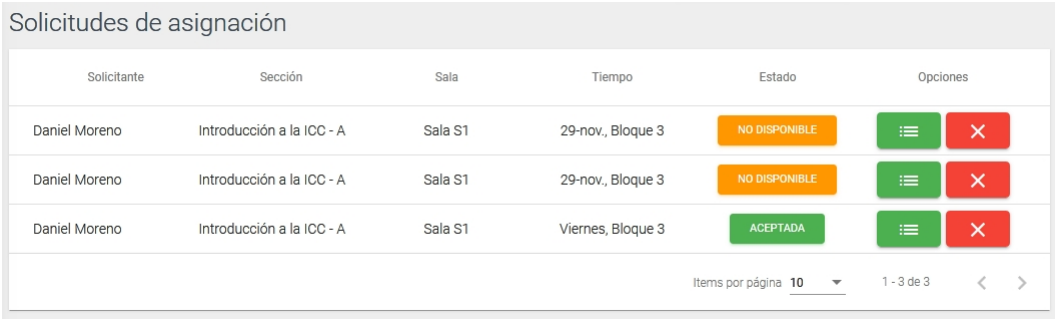


Figura B.24: T11 - Salida esperada de aceptación de solicitud de asignación

B.12. Prueba T12

En la Figura B.25 se muestra que el rechazo de una solicitud se puede realizar directamente desde la tabla de solicitudes. Al rechazar una solicitud se obtiene la salida esperada que implica la eliminación de dicha solicitud ejemplificada en la Figura B.26.

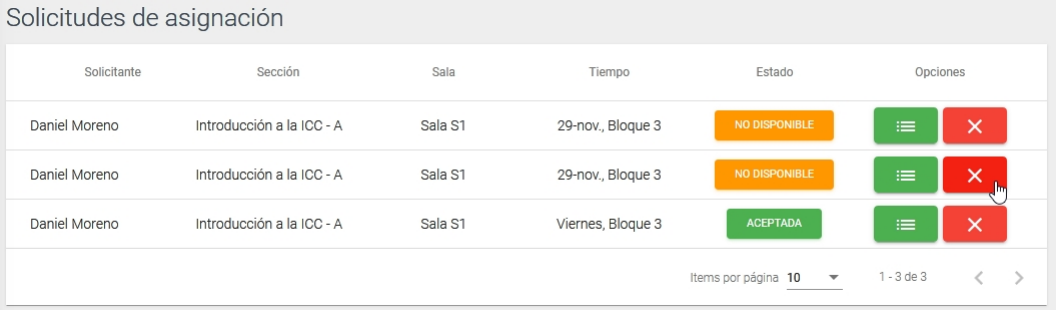


Figura B.25: T12 - Rechazo de solicitud (botón rojo derecho)

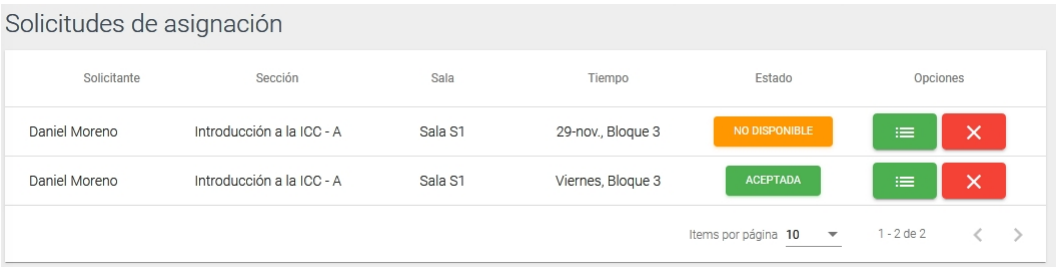
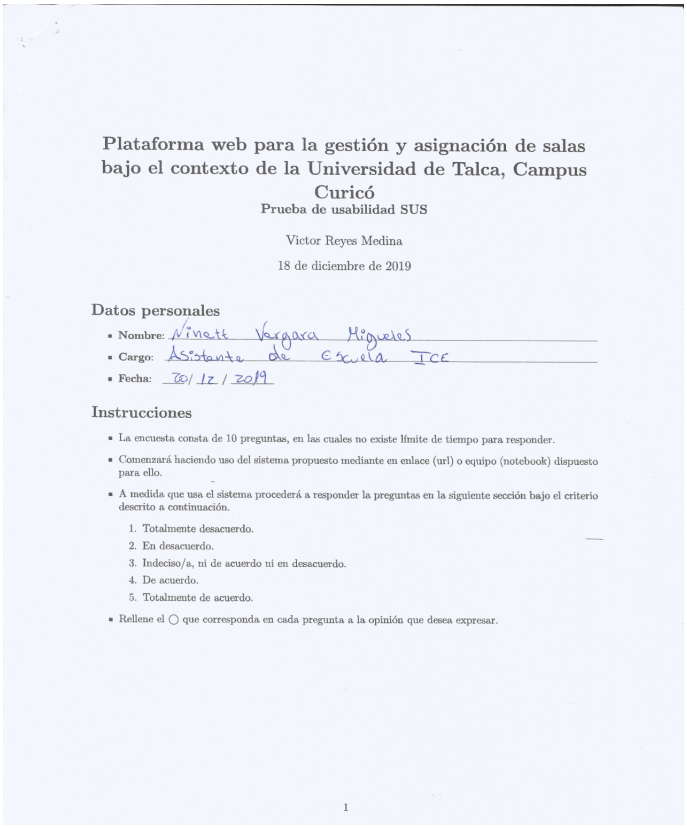


Figura B.26: T12 - Salida esperada, solicitud es eliminada del registro

C. Resultados prueba SUS

A continuación, la imágene en la Figuras 5.5 muestra la primera parte de la encuesta de usabilidad aplicada.



Plataforma web para la gestión y asignación de salas bajo el contexto de la Universidad de Talca, Campus Curicó

Prueba de usabilidad SUS

Victor Reyes Medina

18 de diciembre de 2019

Datos personales

- **Nombre:** Ninett Vargara Higueles
- **Cargo:** Asistente de escuela ICE
- **Fecha:** 20/ 12 / 2019

Instrucciones

- La encuesta consta de 10 preguntas, en las cuales no existe límite de tiempo para responder.
- Comenzará haciendo uso del sistema propuesto mediante en enlace (url) o equipo (notebook) dispuesto para ello.
- A medida que usa el sistema procederá a responder la preguntas en la siguiente sección bajo el criterio descrito a continuación.
  1. Totalmente desacuerdo.
  2. En desacuerdo.
  3. Indeciso/a, ni de acuerdo ni en desacuerdo.
  4. De acuerdo.
  5. Totalmente de acuerdo.
- Rellene el ○ que corresponda en cada pregunta a la opinión que desea expresar.

1

Figura C.1: Prueba de usabilidad - Hoja 1