Universidad de La Habana
Facultad de Matemática y Computación

Sistema para la gestión de horarios.

Autor: **José Carlos Hernández Piñera**

Tutor: **Lic. Pedro Quintero Rojas**

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencias de la Computación

Noviembre de 2022

Agradecimientos

Opinión del tutor

Resumen

Abstract

Índice general

Índice de figuras

Índice de tablas

Parte I

title

Introducción

Desde tiempos de antaño el hombre se ha visto en la necesidad imperiosa de manejar de la manera más adecuada posible sus recursos, no solo para conseguir prolongar la vida de estos, sino además para contar con una aceptada utilización de los mismos. Uno de los principales problemas que aqueja a la sociedad actual es la dificultad que se presenta para garantizar un óptimo manejo de nuestro *tiempo*.

La denominada *gestión del tiempo* hace referencia a la forma en que cada uno organiza y planifica cuánto invierte en actividades específicas; pasar más horas en la empresa no significa ser más eficiente o productivo; por ello, es fundamental una adecuada gestión del horario en el trabajo, lo que permitiría lograr más con menos esfuerzo. Cuando se aprende a administrar mejor el día a día y las labores cotidianas se mejora la capacidad de concentración, lo que trae consigo un mayor enfoque y por tanto una mayor eficiencia. Gestionar el tiempo nos permite realizar las tareas con más rapidez y que la jornada laboral sea más efectiva y se aproveche mejor.

En muchos escenarios cotidianos la planificación de las actividades se realiza solamente de manera irreal, por otro lado, existen ámbitos donde se hace necesario garantizar una manipulación detallada de todo el proceso.

Con la llegada de las tecnologías actuales y la facilidad con que las personas cambian de actividad, el desarrollo de un software que garantice una planificación de nuestra agenda diaria, resulta casi imprescindible.

Un ejemplo claro donde sin duda se comprueba el proceso antes expuesto, es la gestión de las actividades de un centro de estudios y más aún en esta época donde contamos con grandes instituciones universitarias que albergan a cientos de estudiantes con diferentes planes de trabajo y manejan una gran cantidad de recursos y útiles.

Quizá muchos consideren que realizar la labor de confección de un horario para una institución universitaria es algo sencillo, pero cabe destacar que la persona encargada de dicha labor, emplea un valioso tiempo en esto; puesto que hay que preveer que no existan colisiones entre los turnos, los locales, los profesores, además de garantizar que todas las actividades se realicen con la frecuencia que corresponde para garantizar el aceptado desarrollo del proceso de aprendizaje por parte del estudiantado.

Resulta llamativo como un gran número de universidades cubanas hoy en día aún no cuentan con un sistema propio (o de terceros) para realizar las tareas de

confección del horario; sino que en la mayoría de los casos usan softwares como *Microsoft Excel* para gestionar el mismo; lo cual, sin lugar a dudas es sumamente ineficiente y genera un esfuerzo extra para la persona que está desarrollando el mismo, quién sin lugar a dudas debe tener otra serie de tareas que requieran de más importancia.

El trabajo en cuestión persigue presentar un software para proporcionar un sistema que permita manejar el sistema de turnos de clases de la *Facultad de Matemática y Computación* de *La Universidad de La Habana*.

0.1. Motivación

Desde hace poco más de 10 años en la Facultad de Matemática y Computación se ha venido gestado la idea de presentar un modelo de horario propio que sea capaz de satisfacer las necesidades internas de la misma. Con anterioridad se han realizado otras tesis de diplomas dedicadas a abordar cuestiones más especifícas relacionadas con esta índole; dígase: manejo de restricciones, generación automática de horarios, entre otros aspectos.

Las herramientas analizadas previamente, que resolvían el problema del horario, en su mayoría presentaban carencias que se consideraron escenciales a la hora de darle solución al asunto que estamos abordando. Dichas carencias se manejaron dentro del software presentado para garantizar que fueran cubiertas y erradicadas en su totalidad.

Los beneficios finales que ofrecerá el software llaman la atención y es sumamente viable tratarlos como aspectos motivadores a la hora de analizar el resultado final. Se contará por ejemplo, con un sistema de restricciones asociado a todas las entidades del sistema y no solo centrado en los turnos de clases y los profesores; aunque es válido notar que casi todas las condiciones impuestas se relacionan con estos campos.

El software posibilitará también contar con un sistema centralizado, alojado dentro del nodo de la facultad, lo que hará posible que se pueda acceder en tiempo real por todos los estudiantes y los profesores. Además esto hará más factible la distribución y la organización interna de la institución, puesto que cualquier cambio dentro del sistema, puede ser analizado inmediatamente por todos los interesados.

La edición y el mantenimiento del software, también se realizará de manera relativamente sencilla, con esto se garantiza que se pueda adicionar en el futuro cualquier otra funcionalidad que se considere necesaria o que aporte algún beneficio a la universidad.

Los profesores son considerados usuarios privilegiados dentro de la aplicación, pues se permite que estos interactuen directamente con los turnos y que muestren sin necesidad de ser administradores, sus inconformidades - por medio de las restricciones - con la distribución presentada.

Para la persona encargada de la creación del horario, un sistema dedicado especialmente a esto, trae consigo una mejora sustancial; pues es posible detectar datos erróneos durante la confección, se puede tener un control detallado de los

profesores, así como de la gestión de locales para poder determinar fácilmente cuáles se encuentran disponibles en un momento y espacio determinado.

En muchos casos es necesario obtener aulas vacías, simplemente para dedicarlas a otras actividad o porque se pretenden usar en otro tipo de eventos. En un gran número de casos esto puede traer un trabajo extra para la persona encargada de realizar tal gestión. El sistema propuesto posibilita que dicha tarea se realice en cuestiones de segundos y sin esfuerzo alguno, debido a que se proporcionan varias vistas o formas de representar el conjunto de turnos que ofrecen una solucióna al problema que se plantea. La distribución por recursos, cubre tal aspecto; pues es posible indentificar todos los locales dentro de la facultad y los horarios en que están ocupados o libres los mismos; además empleando los filtros personalizados se puede hacer incluso más sencilla la tarea.

0.2. Objetivos

Se tiene como **objetivo general** presentar una aplicación web que cubra las carencias y dificultades que se exhiben en la facultad de Matemática y Computación de La Universidad de La Habana con respecto a la gestión, distribución y creación del sistema de horarios. La aplicación debe ser lo más sencilla posible para permitir que todo tipo de usuarios interactúen con ella. Como **objetivos más específicos**, el sistema deberá:

1. Identificar las colisiones presentes entre los turnos, profesores y locales.
2. Mostrar diversas vistas que hagan posible el chequeo de los turnos de la manera más productiva posible (vista de locales, diaria, por semanas, mensual).
3. Visualizar en culquier momento que se considere necesario, no solo el horario completo sino secciones específicas del mismo.
4. Permitir la generación de archivos en formato *Excel*, para ser exportados, y que cuenten con toda la distribución de los turnos de clase.
5. Permitir el manejo de restricciones o condiciones impuestas sobre todas las entidades del sistema y refelejar el cumplimiento (o no) de estas en una sección dedicada.
6. Ofrecer un manejo de los profesores, asignaturas, grupos, locales y demás aspectos relacionados con una institución educacional.
7. Cubrir el manejo de varias facultades a la vez e incluso, llegado a ser el caso de varias universidades.

El presente trabajo se estructura como se detalla a continuación: en el capítulo 1 se mostrará todo lo referente al estado del arte, así como una descripción de los softwares y herramientas previamente analizados, mostrando además la carencia de estos y por consiguiente la necesidad de construir uno propio que

cubra las necesidades internas de la facultad. En el capítulo 2 se presenta una descripción más detallada del problema así como el enfoque de solución ofrecido. Por último en el capítulo 3 se describirán todos los detalles de implementación referentes al software presentado.

0.3. Convenciones de estilos

Para el presente documento, se mostrarán una serie de figuras o diagramas, para facilitar la comprensión del lector. A continuación se presenta una descripción de las convensiones seguidas para construir las mismas.

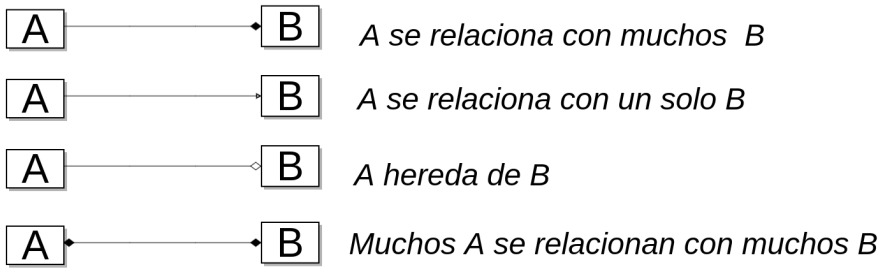

Figura 1: Convenciones de estilos

Capítulo 1

Antecedentes

Mejorar este capitulo.

El tema del manejo, creación y gestión de un sistema de horarios ha sido abordado por un gran número de personas, en muchos ámbitos diferentes.

1.1. Herramientas de gestión

El ejemplo más concreto que se analizó antes de la creación del presente trabajo fue un sistema desarrollado en 2019 entre un grupo de universidades de América del Norte y Europa: UniTime. Este sistema ofrece soporte en un gran número de escenarios, dígase por ejemplo la creación de una especie de salas para la planificación de eventos así como el manejo de secciones individuales por estudiantes. Además cuenta con una pequña comunidad que brinda soporte al mismo por lo que se realizan versiones y modificaciones periódicamente.

Otra herramienta que también llamo la atención fue Unit. Este software confecciona automáticamente los horarios del colegio a partir de los criterios pedagógicos que se determinen en el centro. Propone al usuario una serie de alternativas diferentes sobre las que se pueden hacer cambios manuales. Incorpora además otras funcionalidades como la posibilidad de crear horas de entrada y recreos de los alumnos a horarios diferentes, rangos de horas en que los profesores deben impartir clases por contrato, horario óptimo de guardias, planificación de sustituciones o estadísticas, entre otras. Este generador de horarios está completamente integrado en la plataforma de gestión Alexia.

Otro sistema que resaltó entre los revisados fue iPlanner. Se trata de una lista de tareas que dispone de los ajustes necesarios para elaborar y organizar un itinerario escolar con vista diaria, semanal, mensual y anual que, además, incluye un registro de todos los días festivos locales. Los eventos se dividen en categorías e integra herramientas como informes, compatibilidad horaria en otros países y otras basadas en el diseño del horario como diferentes plantillas,

colores, símbolos y emoticonos. El usuario contará con la opción de facilitar el horario a otros compañeros o estudiantes a través del correo electrónico o subirlo a la nube.

La principal cuestión que motivó el desarrollo de un sistema propio de la universidad y la no utilización de estas herramientas que ya existían fue la posibilidad de ofrecer un manejo de restricciones sobre todas las entidades del horario; restricciones que ofrecen un grado de *felicidad* al ser evaluadas y permiten la apreciación por parte del creador del horario de que tan bueno resulta la distribución brindada a los turnos de clases; que viene siendo, en definitiva, el punto central de todas estas herramientas.

Otros autores manejan en sus sistemas un concepto similar al de *restricción*, pero estas están en todos los casos relacionadas con un profesor y un turno de clase; en cambio en el presente software se permite asociarlas a cualquier entidad definida dentro del sistema, dígase por ejemplo: *local*, *departamento*.

1.2. Restricciones

La idea detrás del manejo e implementación de las restricciones surge a través del trabajo de diploma desarrollado por el estudiante *Joel Rey Travieso Sosa* perteneciente a la Facultad de Matemática y Computación de La Universidad de La Habana.

El análisis y la gestión de las restricciones trajo consigo la clasificación en diferentes grupos:

- *Asignación de recursos*: Cuando un recurso debe ser asignado a otro recurso de distinto tipo o a cierto evento
- *Asignación de tiempo*: Cuando un evento o recurso debe asignarse a una fecha
- *Restricciones de tiempo entre eventos*: Cuando un evento mantiene una relación de tiempo con otro
- *Solapamiento de eventos*: Cuando un grupo de eventos comparte un mismo intervalo de tiempo.
- *Coherencia entre eventos*: Intentar producir horarios más organizados y convenientes
- *Capacidad*: Algunos de los recursos tienen capacidades de uso que no pueden ser vulneradas.
- *Continuidad*: Cuando se intenta producir horarios con determinadas características constantes o muy predecibles.

Además se detectaron cinco grupos fundamentales para clasificar las condiciones o requerimientos del problema:

- *Restricciones unarias*: aquellas que involucran un sólo evento, como por ejemplo, las clases de un curso no pueden ser programadas un día lunes.
- *Restricciones binarias*: aquellas que involucran dos eventos. Un ejemplo típico son las restricciones de topes de horarios para un curso que requiere un mismo recurso: profesor, sala de clases, etc
- *Restricciones de capacidad*: las que se imponen al asignar cursos a salas de clase con capacidad suficiente.
- *Restricciones de separación de eventos*: aquellas que requieren que las actividades estén separadas o siguiendo algún patrón en el tiempo. Algunos ejemplos son las impuestas por políticas de la institución de respetar asignaciones de horarios en patrones predefinidos o las condiciones de no existencia de horas intermedias vacías.
- *Restricciones asociadas a los agentes*: las limitaciones en los horarios asignados para cumplir con las preferencias de los profesores.

En la implementación que se ofrece se manejan 4 tipos de restricciones fundamentales, que cubren la descripción antes mencionada.

- Restricción de requerimiento de cuenta simple.
- Restricción de requerimiento de cuenta de condiciones.
- Restricción de requerimiento de distribución de atributos.
- Restricción de requerimiento relacional.

En los capítulos siguientes se ofrecerá una explicación más detallada de todo los relacionado con estas condiciones impuestas sobre el sistema así como la adecuada definición y formulación de todos los conceptos antes expuestos.

1.3. Tecnologías

Mejorar esto.

Para la confección del sistema se analizaron diversos escenarios para considerar el que propiciara la obtención de un mejor producto y a la vez que se contara con los conocimientos suficientes en el área para obtener los mejores resultados.

El sistema está desarrollado en *NestJS* (framework de NodeJS), usando *VueJS* para el frontend y *PostgreSQL* par el manejo de la base de datos.

En el capítulo 3 dedicado a los detalles de implementación se ofrecerá una explicación más en detalles del porqué de la selección y las bondades que ofrecen las tecnologías empleadas.

Capítulo 2

Enfoque de solución

En el capítulo se pretende hacer alusión a las cuestiones relacionadas con la propuesta de software que se ofrece, brindando definiciones, conceptos, gráficos y demás aspectos que se hagan necesarios para lograr una adecuada comprensión de la implementación realizada.

Se ofrece una solución lo más simple posible, pero que a la vez cubra todas las necesidades y pueda manejar de la manera más adecuada todos los casos de uso que se presenten. Para el desarrollo de la aplicación se emplea un enfoque DDD (Domain Driven Design) y el uso del patrón CQRS (Command and Query Responsability Segregation); en el próximo capítulo (3) se ofrecerá una descripción detallada de tales conceptos así como la razón de la elección.

Si se decidiera describir el sistema en pocas palabras, se pudiera decir que el horario es una colección de turnos de clase y que un turno de clase está asociado a: un profesor, un local, una asignatura, un grupo, una fecha y un tipo (conferencia, clase práctica, laboratorio, etc); además de cada uno de estos elementos tiene sus propios atributos y relaciones entre ellos. Cada profesor puede definir restricciones sobre el sistema y el cumplimiento (o no de estas) se refleja en la felicidad del horario.

El usa de esta herramienta y del sistema de horarios en general necesita de un fujo de trabajo que permita en cada momento contar con los datos necesarios para efectuar los pasos correspondientes.

- Introducción de los datos administrativos: se actualiza el listado de profesores, locales, asignaturas, grupos, facultades y universidades.
- Definición de prioridades dentro de los profesores: a la hora de crear un profesor(o en la edición de este) se hace posible agregar(modificar) el entero que representa la prioridad del mismo. Por defecto la prioridad de un profesor se fija en 1.
- Definición de restricciones: los profesores o el administrador del sistema, pueden agregar condiciones sobre la distribución del horario. Para cada restricción además se define la prioridad que tiene.

- Generación de reportes: obtención de un archivo en formato *Excel* que haga posible la distribución del sistema y la posible visualización del mismo de manera off-line.
- Gestión de semestres y semanas: habilidad de definir los semestres, dígase fecha de inicio y finalización del mismo. Las semanas se crean de forma automática una vez definido el semestre.
- Visualización en diversas modalidades: posibilidad de obtener vistas agrupadas por diferentes condiciones; dígase: vistas diarias, semanales, mensuales y por recursos.
- Modificación, eliminación y edición de los turnos de clase: tarea principal del sistema.

2.1. Entidades del sistema

Para lograr una mayor comprensión y garantizar por tanto que sea entendido el proceso de modelación del problema; se hace necesario presentar, en primera instancia, una descripción de cada una de las entidades que se usan, así como breves datos de utilidad acerca de estas.

Universidad: Garantizar que el software presentado sea capaz de funcionar en el mayor número de escenarios. Esta entidad es considerada la raíz, en el grafo que representa la relaciones entre todas; es decir para describir cualquier gestión o para proceder a la confección del horario se hace necesario la previa definición de la universidad. Es válido notar, que en la mayoría de los casos se manejará solamente una universidad.

Facultad: Maneja todas las facultades presentes dentro de la universidad. La facultad es la encargada de manejar las carreras, así como los locales del sistema. Cada departamento está asociado a una facultad.

Carrera: Agrupa todas las carreras que puedan existir dentro de una facultad. Ofrece una forma de manejar además conjuntos de grupos, profesores y asignaturas.

Departamento: Es la forma más general de agrupar los conjuntos de profesores. Cada departamento pertenece además a una facultad específica. Cada departamento está contenido dentro de una facultad.

Figura 2.1: Relación entre Universidad, Facultad, Carrera y Departamento

Asignatura: Constituye uno de los elementos principales del turno de clases, quién es en definitiva la entidad principal del sistema. Las asignaturas se muestran agrupadas por carreras. Además de cada asignatura se conoce el profesor principal que la imparte.

Grupos: Entidad que se encarga de encapsular conjuntos de estudiantes. Mantiene una relación directa con turno de clase, así como con carrera.

Semestre: Define los diferentes semestres que hay en el sistema. La entidad se llama "semestre" por convención, pero es configurable la cantidad de semanas de duración, mediante una fecha de inicio y una fecha de culminación. En cada semestre se pueden estar gestionando un horario diferente. La idea es que cada usuario del sistema tenga acceso al horario del semestre actual, y que el horario del próximo semestre se pueda ir generando simulatáneamente. Cada semestre está compuesto por varias semanas.

Semana: Esta entidad define y se asocia con una semana del año. Cada semana está contenida dentro de un semestre. Las semanas son generadas automáticamente a partir de la definición del semestre.

Tipo de actividad: Cada uno de los turnos de clase tiene un tipo que resulta importante sobre todo para determinar que tipo de aula se necesita. Los tipos más comunes son las conferencias y las clases prácticas. Aunque en algunas ocasiones se puede definir además actividades de tipo laboratorio. Es posible generar tipos de actividades nuevas, en caso de que se considere necesario.

Profesores: Es una de las entidades de mayor importancia dentro del sistema. Ofrece una vía para asegurar el manejo de todo lo referente a los profesores del centro. Son considerados usuarios especiales o de más relevancia, pues pueden interactuar directamente con el sistema de horarios, imponiendo restricciones o condiciones sobre el mismo. Todos los profesores están agrupados por departamentos. Además cada profesor, puede establecerse como responsable de una asignatura y es posible definir prioridades sobre estos.

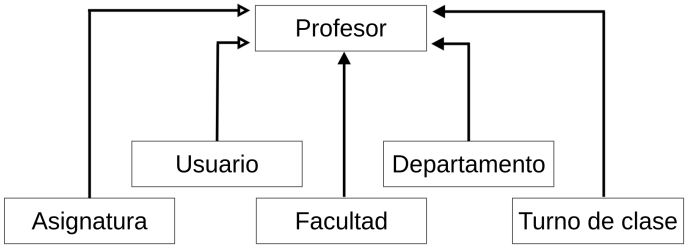

Figura 2.2: Relación de profesor con usuario, asignatura, facultad, turno de clase y departamento

Local: Elemento importante que posibilita conocer el lugar donde se impartirá el turno de clase. Es, junto a asignatura y profesor, uno de los elementos principales de un turno de clase. Por lo general a todos los turnos de clases se les asignará un local. En nuestras universidades existen diferentes tipos de locales: salones de conferencia, aulas, laboratorios, anfiteatros, etc. En este sistema los locales se modelaron pertenecientes a una facultad. Cada uno de los locales se define a partir de un "tipo de local" y esta información es utilizada en algunos de los algoritmos que se encargan del manejo de las restricciones.

Turno de clase: Es la entidad fundamental del sistema. Maneja todo lo

relacionado con las lecciones que se impartirán en cada uno de los días del semestre, por su imporancia se dedicará la sección 2.3 al tratamiento de esta entidad en específico.

Restricciones: Entidad que se encarga de manejar las restricciones o condiciones que los profesores y administradores del sistema pudieran imponer sobre el horario. Las restricciones se describen por medio de varios modelos, uno por cada tipo de restricción manejada. Más adelante se dedicará una sección a la descripción detallada de cada uno de los tipos de restricciones manejados.

Usuarios: Se encarga de gestionar todo lo referente a los usuarios dentro de la aplicación. Está permitido que cualquier usuario realice registro dentro de la misma, pero solamente el administrador del sistema puede determinar cuáles usuarios son profesores y por tanto, que posean un permiso específico de creación de restricciones. Para un usuario registrado sin permisos, solo es posible la visualización del sistema, cerrando toda forma de modificación o creación de nuevas entidades dentro del mismo. Un usuario puede estar relacionado, a lo sumo, con un profesor.

Reportes: Módulo que posibilita la obtención de un documento en formato *Excel* que puede ser útil para la posterior distribución del horario, o para la revisión del mismo de manera off-line. El documento presenta una serie de páginas: una por cada grupo descrito en el horario y una extra que meneja una representación por locales, es decir, la disponibilidad de los mismos durante todo el semestre. En cada página se encuentra descrito el horario por semanas físicas y además se ofrece una leyenda con la descripción de los turnos de clase así la hora de comienzo y finalización de los mismos.

Cada entidad posee un conjunto de atributos base; por medio de la herencia se logra plasmar esto a la hora de realizar la implementación.

Todas las entidades del sistema heredan de `DomainEntity`, que no es más que una *clase abstracta* con un constructor privado y una definición específica de un método *equals*. Esta clase base cuenta además con las propiedades base antes mencionadas. Dichas propiedades son:

- *Nombre completo*: Propiedad de tipo texto que almacena el nombre en formato largo del objeto que se esté referenciando en ese momento.
- *Nombre reducido*: Propiedad de tipo texto que almacena el nombre en formato corto del objeto que se esté referenciando en ese momento.
- *Descripción*: Propiedad de tipo texto que contiene una descripción, que se considere conveniente presentar, acerca del objet que se esté referenciando en ese momento. Esta pripiedad en puede tomar valores nulos.
- *Prioridad*: Propiedad de tipo numérico que almacena la prioridad que pudiera tener un objeto (o entidad) sobre otro en el sistema. Esta propiedad es utilizada principalemte en el cálculo de la *felicidad del sistema*.

2.2. Modelo entidad-relación (MERX)

2.3. Turno de Clase

El *turno de clases* es la entidad fundamental que compone todo sistema de horarios. Como aspecto común a la mayoría de los turnos de clases en universidades cubanas se tiene que estos tienen una duración de 90 minutos y que se pueden clasificar en dos tipos básicos: conferencias y clases prácticas.

En la Universidad de La Habana se estila a separar el día en dos secciones: mañana y tarde y que cada sección esté compuesta por tres turnos de clases, es válido notar que esto se realiza por una especie de convenio ya preestablecido y que el sistema cuenta con las herramientas para variar este funcionamiento y posibilitar, por tanto, la modificación de los mismos.

Un turno de clases está compuesto por los siguientes atributos:

- *Atributos base*: definidos en la sección 2.1
- *Inicio*: marca la hora de inicio del turno de clase.
- *Finalización*: marca la hora de finalización del turno.
- *Id de recurso*: referente al `id` del local; se utiliza mayormente en el manejo de la vista asociada a los recursos.
- *Semana*: referente a la semana de creación del turno de clase.
- *Profesores*: colección de elementos que representan los profesores encargados del turno de clases; notar que al menos siempre. se contará con un profesor responsable, que se toma de la asignatura referente al turno.
- *Local*: referente al local donde se impartirá el turno de clase.
- *Asignatura*: referente a la asignatura.
- *Tipos de clase*: referente al tipo de clase (conferencia, clase práctica, laboratorio).
- *Grupo*: referente al grupo.

Es común notar que en muchos sistemas de horarios, por no decir en todos, contamos con la misma frecuencia de turnos todas las semanas, o a lo sumo en intervalos de quince días. Para ganar en usabilidad y por tanto mejorar la experiencia de usuario se presenta un enfoque que permite crear turnos en una frecuencia específica; dicho en otras palabras y mostrado a través de un ejemplo: si tenemos, dado el caso, que el turno de Matemática Discreta de 2^{do} de Computación se realiza los martes de todas las semanas, en la aplicación no es necesario recorrer todos los martes del semestre e ir creandolos uno por uno, pues se ofrecer una especie de configuración y el sistema se encarga de la posterior creación de cada uno. De cualquier forma también es posible la creación de turnos individuales sin necesidad de crearlos en serie.

El manejo de turnos en serie nos crea una relación entre los mismos, o sea, los turnos de una misma serie están relacionados entre ellos, esto hace posible que la edición y eliminación de cualquier turno de la serie influya también en el resto de estos turnos. Estas operaciones son además flexibles, dado el hecho de que también es posible modificar inidividualmente los turnos de cualquier serie sin afectar al resto.

El sistema cuenta con una serie de filtros dedicados a obtener secciones del mismo. Los turnos de clase están adaptados, por tanto, a estos requerimientos. Se hace posible agregar filtros por tiempo (día-hora de inicio y finalización), así como por asignaturas, grupos, locales y tipos de clase.

2.4. Manejo de restricciones

El manejo y gestión de restricciones es un importante *feature* ofrecido en el sistema. Cada profesor propone un conjunto de condiciones que se aplica a los turnos de clase, además de cada profesor se conoce la prioridad, así como la prioridad particular que le otorga a cada restricción. Con el objectivo de expresar un concepto un poco más formal:

Definición 2.4.1 (Restricción) *Forma por medio de la cuál los usuarios (principalmente los profesores) expresan sus consideraciones acerca de los elementos que determinan la calidad de cualquier horario propuesto; ya sea en sentido general o personal.*

Cada restricción cuenta con un grupo de elementos comunes (figura 2.3):

Condiciones. Permite identificar a qué turnos se exigirán determinados requerimientos. Define el filtro inicial que se impone sobre el conjunto de turnos, para obtener así un subconjunto de los mismos, los cuáles deberán cumplir estrictamente los requisitos planteados.

Una condición está compuesta por un atributo `A`, y en función de este, un operador `O` y un valor `V`. Un atributo relativo a un turno puede ser en realidad, un atributo relativo a una entidad asociada al turno de clase.

Ejemplo	**Atributo**	**Operador**	**Valor**
Aula con más de 30 alumnos	Capacidad del Local	>	30
Ser un martes	Día del Turno	==	martes

Tabla 2.1: Ejemplos de representación de condiciones

Es posible también garantizar el manejo de grupos de condiciones para asegurar que se puedan formular la totalidad de las operaciones deseadas.

Definición 2.4.2 (Grupo de condiciones) *Forma amigable de representar un bloque de condiciones lógicas encerradas entre paréntesis; y asegurar, por tanto, la manera correcta de evaluación del mismo. El conjunto de condiciones de una restricción puede verse como un grupo de condiciones.*

Ejemplos de grupos de condiciones:

- "Ser lunes". Una condición que involucra un solo operador lógico también es considerado un grupo de condiciones.
- (Turnos de Matemática Discreta) (del martes).
- (Turnos de Matemática Discreta o del Matemática Numérica) (del martes).

Intervalo. Ofrece una forma de obtener una separación del conjunto de turnos en subconjuntos, cada uno conteniendo los turnos correspondientes a un período de tiempo diferente, en cada caso de tamaño *intervalo*, salvo, quizás, el último, en caso de que el curso no pueda ser dividido exactamente en un número entero de intervalos de tamaño *intervalo*. El objetivo detrás de esta separación del conjuto de turnos en intervalos, en segregar aún más la evaluación; pues, por ejemplo, supogamos que estamos analizando una restricción sobre X y que: $\exists y \in X$ tal que para y la restricción se incumple, entonces resultaría un poco antiintuitivo decir que la restricción se incumple, solo porque fallara en un turno, por ello si consideramos que interval $= 7$ (análisis semanal), entonces el turno entraría solo en una semana de las analizadas y por tanto la restricción no se marcaría en su totalidad como incumplida, lo que influiría positivamente en la felicidad del sistema.

Prioridad. Establece la prioridad de la restricción. El usuario creador de la restricción (profesor o administrador del sistema) deberá establecer la priridad o cuán importante es la restricción que se está definiendo. Por defecto, si el valor no se proporciona, se fija en 1. Este valor se utilizará posteriormente para el cálculo de la *felicidad* del sistema. La utilidad del mismo radica en gestionar de alguna manera una especie de jerarquía y posibilitar, por tanto un orden, para lograr ser lo más semenjante posible a la vida real.

Descripción. Este campo se presenta de manera obligatoria a la hora de la creación de nuevas restricciones. Es un forma de representar con palabras lo que se pretende crear y hacer posible por tanto la rápida comprensión e interpretación de la misma.

Profesor. Se emplea para referenciar al profesor al que está destinada la restricción. En cada de que la restricción sea creada por el profesor en sí, y no por el administrador del sistema, el valor del campo se fijará automáticamente.

El conjuto de campos definidos por cada tipo de restricción varía en dependencia del tipo.

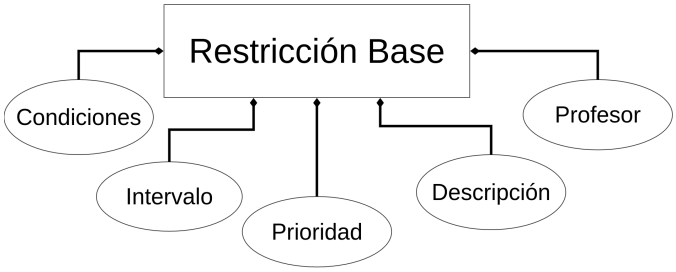

Figura 2.3: Restricción base.

2.4.1. Tipos de restricciones

Es posible manejar o declarar diversos timpos de restricciones, dichos tipos ya fueron enunciados en las secciones previas (1.2) de este documento.

Restricción de requerimiento de cuenta simple. Esta restricción se refiere al número de turnos que han satisfecho las condiciones fijadas inicialmente. El usuario debe configurar el operador de comparación O a aplicar y el valor V con el cual comparar. La restricción se cumple si la cantidad de turnos que satisfacen las condiciones mantiene una relación O con V. El valor de V puede ser numérico, porcentual o bien fraccionario con respecto al total de turnos que pasan la etapa de las condiciones.

Restricción de requerimiento de cuenta de condiciones. Este requerimiento se refiere a qué parte del conjunto de turnos que ha satisfecho las condiciones previas, cumple también un bloque de condiciones adicionales, y cuenta además con el operador de comparación O a aplicar y el valor V con el cual comparar. El requerimiento se cumple si la cantidad de turnos que satisfacen las condiciones iniciales y satisfacen simultáneamente otras que se han configurado, mantiene una relación O con V.

Restricción de requerimiento de distribución de atributos. Este requerimiento se refiere a la cantidad de valores distintos que puede tomar determinado atributo en el conjunto de turnos que pasa la etapa de condiciones. El usuario configura, además del atributo A, el operador de comparación O y el valor V. El requerimiento se cumple si la cantidad de valores que toma el atributo A en el conjunto de turnos que cumple las condiciones previas mantiene una relación O con V. En este caso el valor de V debe ser numérico.

Restricción de requerimiento relacional. Este requerimiento se refiere a la relación que mantiene el conjunto de turnos que cumplen las condiciones previas con otro que cumple otro grupo de condiciones. En este caso se requiere que el usuario añada un segundo grupo de condiciones que filtrarán el segundo conjunto de turnos, un atributo A y un operador booleano de conjuntos O que define la relación entre los valores de A en el primer conjunto y el segundo. El requerimiento se cumple si el conjunto de valores que toma A en el primer conjunto de turnos mantiene una relación O con el conjunto de valores que toma A en el segundo conjunto.

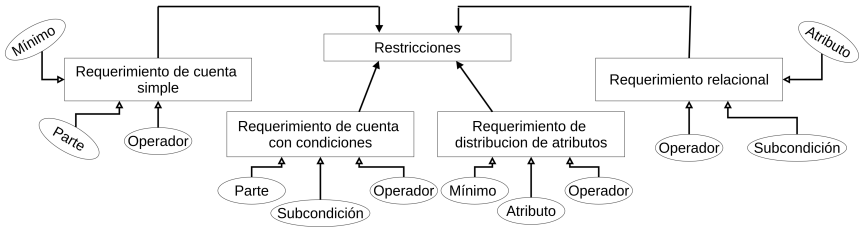

Figura 2.4: Restricciones con sus atributos.

2.4.2. Felicidad del sistema

La felicidad del sistema, es una cuestión a la que se ha venido haciendo alusión desde los capítulos anteriores. Este valor se obtiene a través de la felicidad relativa a cada uno de los profesores y su posterior procesamiento teniendo en cuenta una fórmula previamente definida.

Definición 2.4.3 (Felicidad del sistema) *Valor definido entre 0 y 100 que no es más que un medidor de la calidad del horario.*

La estrategia utilizada para arribar a las fórmulas de cálculo de felicidad en profesores y en el sistema completo, es básicamente la misma; y se basa en el peso de las prioridades (2.4) establecidas con anterioridad en cada unas de las entidades.

2.4.3. Fórmula de la felicidad relativa a un profesor

Para calcular la felicidad relativa a un profesor con respecto al cumplimiento de sus restricciones, se empleará la siguiente fórmula.

$$f_p = \frac{w_{1p}c_{1p} + \cdots + w_{np}c_{np}}{w_{1p} + \cdots + w_{np}} \tag{2.1}$$

Donde se tiene que:

- $w_{ip} \in \mathbb{N}_+$: Prioridad del requerimiento i-ésimo del profesor p-ésimo

- $c_{ip} \in \{0,1\}$: Cumplimiento del requerimiento i-ésimo del profesor p-ésimo
- $n \in \mathbb{N}_+$: Cantidad de requerimientos
- $f_p \in [0,1]$: Felicidad del profesor p-ésimo

2.4.4. Felicidad relativa al sistema

Para calcular la felicidad total del sistema se emplea el cómputo de las felicidades respectivas de los profesores:

$$F_T = \frac{f_1 p_1 + \cdots + f_m p_m}{p_1 + \cdots + p_m} \tag{2.2}$$

Donde se tiene que:

- $f_p \in [0,1]$: Felicidad del profesor m-ésimo.
- p_m: Prioridad del profesor m-ésimo.
- $m \in \mathbb{N}$: Cantidad de profesores.

2.4.5. Arquitectura

El sistema presentado, no es ni mucho menos sencillo, encierra una gran lógica de negocio y encontrar una forma aceptada de representar el mismo puede resultar, en muchos casos, de gran dificultad.

Para asegurar el correcto diseño de la aplicación, su posible extensión y adición de nuevas features (características), se hizo neceario utilizar una arquitectura, quizá un poco compleja, pero que permitiera enfocar todos los casos de uso que puedan surgir.

DDD o Domain Driven Design (por su siglas en inglés) es un enfoque para el desarrollo de software con necesidades complejas mediante una profunda conexión entre la implementación y los conceptos del modelo y núcleo del negocio.[1]

El diseño guiado por el dominio (DDD) propone un modelado basado en la realidad de negocio con relación a sus casos de uso. En el contexto de la creación de aplicaciones, DDD hace referencia a los problemas como dominios. Describe áreas con problemas independientes como contextos delimitados (cada contexto delimitado está correlacionado con un microservicio) y resalta un lenguaje común para hablar de dichos problemas. También sugiere muchos patrones y conceptos técnicos, como entidades de dominio con reglas de modelos enriquecidos (no modelos de dominio anémico), objetos de valor, agregados y raíz agregada (o entidad raíz) para admitir la implementación interna.[2]

DDD es capaz de adaptarse a diversos tipos de arquitectura, para el software que nos envuelve se emplea una arquitectura por capas (Layered Architecture). Este enfoque es, sin lugar a dudas, una de los más empleados hoy en día. Esta impone una estructura y un acceso restringido a cada capa; dígase: dominio, aplicación, infraestructura y presentación. Las capas surgen para garantizar la separación de las responsabilidades, en otras palabras: cada una se encarga

de gestionar cosas específicas. Una capa superior puede acceder a las capas inferiores, pero a la inversa no es posible.

Muchos autores consideran o relacionan este enfoque con Arquitectura Hexagonal (Hexagonal Architecture), también conocida como: Puertos y Adaptadores; donde los puertos son interfaces y los Adaptadores son las implementaciones de estas. A modo práctico, se puede entender como una aquitectura por capas donde el principo de Inversión de Dependencias fue aplicado, por lo que, la capa de dominio se vuelve la capa central del sistema y no "depende"de ninguna otra capa.

- Dominio: Gestiona todo lo referente a la lógica que involucra las entidades descritas en las secciones previas (2.1). Es la capa fundamental dentro de DDD.
- Aplicación: Se encarga de manejar todos los casos de uso que relacionados con cada entidad. A modo de ejemplo, todo sistema cuenta con 4 casos de uso básicos, por así llamarlos: crear, leer, eliminar, editar.
- Infraestructura: Maneja todo lo relacionado con el acceso a datos. Contiene los repositorios y por tanto define una especie de envoltura (wrapper) que evita que se acceda directamente a la base de datos.
- Presentación: Se encarga de exponer todos los puntos de acceso a la API (Application Program Interface), así como los consumidores (consumers) que se encargan de interactuar con aplicaciones de terceros.

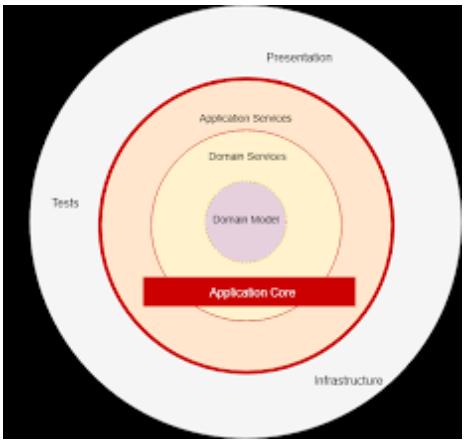

Figura 2.5: Domain Driven Design

Los principios *SOLID* abordados por Robert C. Martin (autor de diversos libros como Clean Code y Clean Arquitecture) se interaron seguir a través de toda la implementación. No resulta llamativo, comentar acerca de SOLID en este documento, solamente a modo de referencia se pretende mostrar los nombres detrás de SOLID.[3]

- **S**: Single-Responsibility Principle.
- **O**: Open-Closed Principle.
- **L**: Liskov Substitution Principle.
- **I**: Interface Segregation Principle.
- **D**: Dependency Inversion Principle.

Capítulo 3

Detalles de implementación

Con el objetivo de garantizar que el lector esté más acorde con la tecnología usada para la implementación, las razones de esa elección, la forma de desplegar el sistema y demás aspectos relacionados con la parte tecnológica se expone esta capítulo.

A día de hoy se cuenta con muchas herramientas para desarrollar un software de este tipo. En esta ocasión se escogión una tecnología que ha alcanzado bastante auge en los últimos años. En las secciones siguientes se hará alusión a la misma así como al lenguaje empleado para el desarrollo.

3.0.1. Lenguaje de programación

Para este proyecto se utilizó el lenguaje TypeScript, auxiliándonos de NestJS (framework de NodeJS).

TypeScript

[4]

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases. Anders Hejlsberg, diseñador de C# y creador de Delphi y Turbo Pascal, ha trabajado en el desarrollo de TypeScript. TypeScript es usado para desarrollar aplicaciones JavaScript que se ejecutarán en el lado del cliente o del servidor, o extensiones para programas (Node.js y Deno).

TypeScript extiende la sintáxis de JavaScript, por tanto cualquier código JavaScript existente debería funcionar sin problemas. Está pensado para grandes proyectos, los cuáles a través de un compilador de TypeScript se traducen a código JavaScript original.

TypeScript soporta ficheros de definición que contengan información sobre los tipos de librerías JavaScript existentes; esto permite a otros programas usar

los valores definidos en los ficheros como si fueran entidades TypeScript de tipado estático. El compilador de TypeScript está escrito asimismo en TypeScript.

El lenguaje fue publicado en octubre de 2012 , después de dos años de desarrollo por parte de la compañía y desde esa fecha ha tenido en general, buena aceptación por parte de todos lo que le mereció el mérito en 2020 como segundo lenguaje de programción más amado según la encuesta de Stack Overflow[5] 2020 Develop Survey.

NestJS

[6] NestJS es un framework para construir eficientes y escalables aplicación del lado del servidor utilizando NodeJS. Posee soporte tanto para JavaScript como para TypeScript y combina elementos de Programación Orientada a Objetos (OOP, por sus siglas en inglés), Programación Funcional y Programación Funcional Reactiva.

Nest posee un robusto framework HTTP basado en *Express* (definido por defecto) y opcionalmente se puede configurar además el uso *Fastify* como alternativa a *Express*.

Nest ofrece un nivel de abstracción superior al que normalmente encontramos en los demás frameworks de Node(Express/Fastify), pero también expone sus APIs directamente al desarrollador. Esto le ofrece a los desarrolladores la libertad de emplear la mayoría de las aplicaciones de 3ros desarrolladas para esos otros frameworks.

En los últimos años gracias a Node, JavaScript se ha convertido en una gran herramienta web tanto para front como para aplicaciones de backend. Se presentan impresionantes proyectos como Angular, React, Vue; los cuáles mejoran la productividad de los desarrolladores y hacen que el desarrollo de nuevos productos sea más rápido, más testeable y más extensible. Por otra parte y a pesar de las bondades que ofrece Node, aún no presenta una solución efectiva al problema de arquitectura.

Nest se presenta como solución a esto, ya que posee una estructura que induce desde el principio a la utilización de patrones arquitectónicos. La arquitectura de Nest es inspirada en Angular.

Fue desarrollado por Kamil Mysliwiec, desarrollador de Google; el lanzamiento de su versión 9 se realizó el 8 de Julio de 2022.

3.0.2. Aplicación visual

La idea detrás de la aplicación visual surge inspirada en un sistema desarrollado hace un par de años dentro de la facultad por un grupo de estudiantes y que se presentó como proyecto a la jornada científica de ese curso.

Se ofrece un sistema desarrollado en VueJS y que pretende seguir las buenas prácticas de desarrollo de frontend. Está escrito en JavaScript.

El sistema posee un conjunto de vistas dedicadas a manejar todas las tareas administrativas referentes al software; dígase: creación de profesores, grupos, semestres, asignaturas y demás cuestiones referentes al centro educacional.

Solamente para el usuario con los permisos adecuados están habilitadas tales características, es decir el usuario administrador es el único que puede realizar modificaciones internas dentro del sistema.

En la página principal se definen 5 tipos de filtros que hacen posible una rápida interacción con el sistema. Estos filtros se presentan útiles en un gran número de escenarios. En la página principal se muestra además la opción de descargar el horario, esta está habilitada para cualquier tipo de usuarios. El número que se presenta en la parte superior izquierda es referente a la felicidad del sistema, aspecto que fue abordado en las secciones previas a este capítulo. (2.4.2)

En la visualización general del horario, cada grupo posee un color específico para que se haga más sencilla su lectura, el color es posible definirlo por el administrador del sistema a la hora de la creación del grupo, luego todos los turnos de clase que se asocien al mismo, se mostrarán de ese color.

En el manejo de la interfaz visual quizá resulte un poco llamativo para el administrador del sistema; que es el que tiene acceso a esas vistas; la creación, modificación y eliminación en serie que se muestra en cada *modal* referente a los turnos de clases. Este aspecto fue abordado con anterioridad, pero la acción que realiza es modificar todos los turnos que posean las mismas características del turno actual, es decir, que repitan el ciclo del horario para él. Un ejemplo de esto se evidencia claramente en: todos los turnos del martes a 4to turno de Matemática Discreta para 2^{do} de Ciencias de la Computación.

Para cada turno visualizado es valido además modificarle la duración del mismo, esto se logra modificando el *size* de este dentro del horario. Realizar esta acción es justo como modificarle el *size* a cualquier otra ventana del sistema, lo que en esta ocasión es al caja que describe el turno.

Haciendo clic encima del turno se obtine además una descripción detallada del mismo, así como las opciones para la edición múltiple.

3.0.3. Autenticación

La autenticación es una parte esencial en la mayoría de los sistemas y aplicaciones. Existe muchos formas y estrategias de manejar la misma. La autenticación dentro del sistema se maneja con un módulo independiente que posee todos los casos de uso y middlewares relacionados con tal aspecto. En esta ocasión el proceso se realiza por medio de JSON Web Token(JWT).

JWT es un estándar de internet propuesto para la creación de firmas y/o encriptación (opcional) cuyo contenido es enviado en forma de cadena de texto de backend a frontend (y viceversa). Los tokens que se generan son firmados usando un protocolo de clave pública y clave privada. La forma de funcionamiento de JWT resalta por su simpleza, el servidor se encarga de generar un token que puede contener información referente al usuario autenticado (en la mayoría de los casos maneja los permisos de los usuario, así como el ID). Este token es enviado del servidor al cliente y este último se encarga de almacenarlo en las cookies o el local storage. Luego el cliente en cada una de las peticiones

adjunta este token y de esa forma el servidor puede identificar correctamente al usuario que pretende acceder a los puntos de acceso definidos.[7]

Cuando se recibe una petición de login por parte del cliente, el servidor se encarga de comprobar que las credenciales son correctas y la respuesta del login, es entre otras cosas, el token al que se hace referencia en el párrafo anterior. Supongamos después que se hace otra petición cliente-servidor y que el token se adjunta de forma apropiada en las cabeceras de la misma, cuando la petición arriba al punto adecuado, el servidor se encarga de ejecutar un *middleware* (función intermedia) que procesa el token y verifica que sea el correcto, si todo este proceso se ejecuta sin mayores contratiempos, entonces en el request de la petición a través del campo user (request.user) se puede acceder al usuario que está intentado realizar la acción. En otro caso se responde con un estado 401 lo que indica que el usuario no esta autorizado al acceder al recurso que intenta solicitar.

Todo este proceso antes descrito se realiza por medio de un paquete de terceros *@nestjs/jwt* y además con la intervención de *passport-jwt*

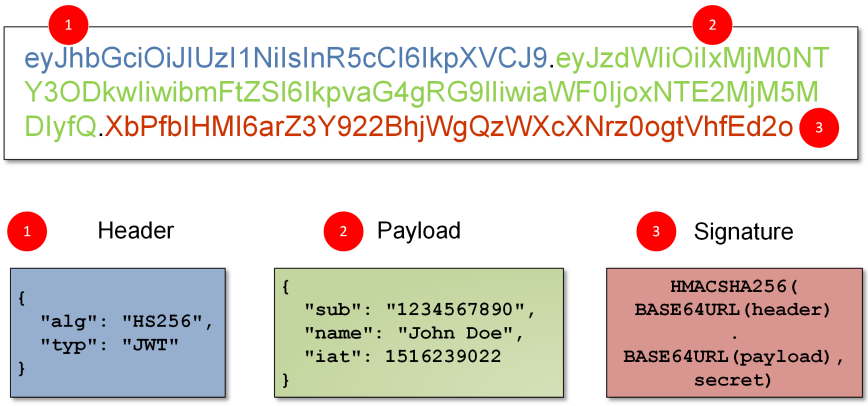

Figura 3.1: Restricciones con sus atributos.

Conclusiones

Referencias

[1] Domain-driven design from wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Domain-driven_design`. 18

[2] Mike Rousos Cesar de la Torre, Bill Wagner. *.NET Microservices: Architecture for Containerized .NET Applications*. 18

[3] Ugonna Thelma. The s.o.l.i.d principles in pictures. 19

[4] Wikipedia. Typescript. `https://es.wikipedia.org/wiki/TypeScript`. 21

[5] Stack Overflow. Stack overflow. `https://stackoverflow.com/`. 22

[6] Kamil Mysliwiec. Nestjs. `https://docs.nestjs.com/`. 22

[7] Wikipedia. Json web token. `https://en.wikipedia.org/wiki/JSON_Web_Token`. 24