Documento de la arquitectura del software

Para

Nitrate
Versión 1.0 aprobada

Preparado por:
Josué Arrieta Salas
Adrián López Quesada
Seth Michael Stalley

3/1/2017

Historial de Revisiones

Nombre	Fecha	Descripción	Versión
Sad Nitrate	3/1/2017	Creación del primer documento de la arquitectura Nitrate. Para la primera iteración de casos de uso.	1.0

Índice

Introducción

En este documento se encontrará la definición de la arquitectura del sistema Nitrate. Primero se hablará sobre las generalidades de la mismo y el propósito y alcance del proyecto. Luego se hablarán de las distintas vistas que posee la arquitectura: casos de uso, diagramas lógicos (de clases y de entidad-relación), diagramas de vistas de proceso y diagramas de despliegue.

Propósito

Este documento provee una guía para la comprensión de la arquitectura de sistema Nitrate, usando un número diferente de vistas sobre la arquitectura para describir diferentes aspectos del sistema. Este documento también está enfocado en capturar y convenir en las decisiones significativas sobre la arquitectura que serán implementadas en el sistema. Esta primera versión de este documento estará enfocada en la primera iteración de casos de uso. Tal documentación, y toda documentación se podrán revisar en referencias.

Es pensado que este documento sea consultado mayoritariamente por los desarrolladores del sistema Nitrate, ya que funcionará como una guía para la implementación de la arquitectura. Será crítico en la toma de decisiones. Se escoge este grupo en específico este grupo debido al alto nivel técnico del documento que posee. Será más de un manejo interno.

Alcance

Este documento de arquitectura de software provee una visión general del sistema Nitrate para Laura Hernández. Este sistema será desarrollado con el propósito de automatizar la obtención de la concentración de nitratos en una muestra de agua para determinar si el agua es apta para el consumo humano. Se planea aumentar la eficiencia y eficacia del proceso.

Para la primera versión del y primera iteración del proyecto Nitrate se pueden listar las siguientes características esenciales:

- La aplicación debe de poder procesar y cargar archivos de texto con los valores de absorbancia por *wavelength* y metadatos de manera automática.
- Se tendrá una carpeta que el programa estará observando, de manera que cada vez que un archivo de texto es ingresado, este archivo es cargado de forma automática. Esta carpeta es seleccionada por el usuario.

- Por cada archivo de texto que fue cargado, se puede seleccionar un valor de desviación estándar (STD) para este.
- Calcular el valor de la concentración de una muestra a partir de una calibración seleccionada.
- Generar la fórmula de concentración con base en la correlación de las desviaciones estándares seleccionadas.

Este documento se generó mayormente del documento de especificación de requerimientos. También es importante mencionar que si se quiere revisar con aún más detalle, se recomienda revisar el documento de Visión y Alcance que se realizó anteriormente. Se puede consultar en: https://drive.google.com/open?id=0Bwn9E8E9d8OwS0Q0VXlSYUV6VTQ . También se puede consultar el Project Charter en el apartado de Alcance del proyecto: https://drive.google.com/open?id=0Bwn9E8E9d8OwWFlXTkVCbEtVUFk. En estos documentos se describen las funcionalidades a la que la arquitectura de este SAD corresponderá de manera totalmente completa.

Definiciones, Acrónimos y Abreviaciones

La definición de términos, acrónimos y abreviaciones utilizadas para la comprensión de este documento se puede encontrar en el glosario en la última sección de este documento: Glosario. También se recomienda revisar para aún mayor comprensión glosario que se encuentra en el ERS y en el plan de pruebas.

Referencias

Anterior a este documento se han realizado los siguientes documentos que se podrían desear consultar. En los mismos documentos se especifican la versión de los mismos:

- Project Charter versión 1.0: https://drive.google.com/open?id=0Bwn9E8E9d8OwWFlXTkVCbEtVUFk .
- Documento de Visión y Alcance versión 1.0: https://drive.google.com/open?id=0Bwn9E8E9d8OwS0Q0VXlSYUV6VTQ .
- Listado de casos de uso: https://drive.google.com/open?id=0Bwn9E8E9d8OwZUEtNURlaFkxZU0 .

- Plan de pruebas versión 1.0: https://drive.google.com/open?id=0Bwn9E8E9d8OwNWZncmRXbklBbDA .
- Estatuto de requerimientos del sistema (ERS) versión 1.0: https://drive.google.com/open?id=0Bwn9E8E9d8OwQXZIWmlUMVM5dDA .

Generalidades

El siguiente documento se encuentra organizado de la siguiente manera:

Sección	Descripción
2	**Representación de la arquitectura** Esta sección describe cuál es la arquitectura del sistema actual y cómo será representaba. Se presentan las distintas vistas que serán utilizadas y se explica en qué consiste y qué incluye cada vista. Si es necesario también se hace una referencia a documentos asociados
3	**Metas y limitaciones de la arquitectura** Esta sección describe los requerimientos de software y los objetivos que tienen un impacto significativo en la arquitectura. También se capturan las limitaciones especiales acerca de la arquitectura del sistema a implementar.
4	**Vista de casos de uso** Esta sección enlista los casos de usos o escenarios del modelo de casos de uso. Estos representan una funcionalidad central del sistema final y tienen un gran impacto en la arquitectura.
5	**Vista lógica** Esta sección muestra los componentes de software significativos de la arquitectura de la aplicación. Se muestran estos cómo interactúan y sus interfaces. Se mostrará utilizando diagrama de paquetes y diagrama de dominio.
6	**Visa de procesos** Esta sección describe la composición del sistema dividida en procesos o *threads*.

	Se ven los distintos flujos de ejecución del sistema.
7	**Vista de despliegue** Esta sección se describe rápidamente las diferentes configuraciones en la re (hardware) en donde el sistema será desplegado. Debido a la razón que el hardware no es parte fundamental del proyecto, se describirán brevemente.
8	**Vista de implementación** Esta sección describe la estructura general de la implementación del modelo. Se descompone en proyecto en capas. Se utilizará un diagrama de componentes.
9	**Vista de datos** Esta sección muestra una descripción de la perspectiva respecto al almacenamiento persistente de datos. Se utilizará el esquema de entidad-relación de la base de datos.
10	**Tamaño y rendimiento** Una descripción de las mayores características del software que impactan la arquitectura, además de limitaciones de rendimientos.
11	**Calidad** Es una descripción de la forma en que la arquitectura del software contribuye a las características del sistema, tales como la extensibilidad, la portabilidad y otras. Además indica si alguna de estas características tiene algún papel importante dentro del sistema.

Representación de la arquitectura

Este documento presenta la arquitectura del sistema como una serie de vistas, de las cuales se pueden mencionar: vista de casos de uso, vista lógica, vista de procesos y vista de implementación. Estas vistas son implementadas mediante modelos UML desarrollados dentro de Visual Paradigm. También se optó por herramientas en línea tales como draw.io para compartir y editar en tiempo real los diagramas. Se tienen las vistas

Vista de casos de uso

- Audiencia: todos los *stakeholders*.
- Área: describe un set de escenarios y/o casos de uso que son críticos para la arquitectura.
- Documentos asociados: ERS y documento de Visión y Alcance.
- Representación utilizada: diagrama de casos de uso, descripción de cada caso de uso.

Vista lógica

- Audiencia: los desarrolladores del sistema Nitrate.
- Área: describe la organización del sistema en distintas agrupaciones lógicas, módulos o paquetes.
- Documentos asociados: no hay.
- Representación utilizada: diagrama de paquetes y diagrama de dominio/clase.

Vista de procesos

- Audiencia: los desarrolladores del sistema Nitrate.
- Área: describe cómo los componentes lógicos se asocian con los procesos del sistema.
- Documentos asociados: no hay.
- Representación utilizada: diagrama de procesos.

Vista de implementación

- Audiencia: los desarrolladores del sistema Nitrate.
- Área: describe los componentes de software y cómo estos están interactúan.
- Documentos asociados: no hay.
- Representación utilizada: diagrama de componentes

Vista de Despliegue

- Audiencia: los desarrolladores del sistema Nitrate.
- Área: describe la relación entre el software y el hardware. Muestra cómo el sistema está distribuido.
- Documentos asociados: no hay.
- Representación utilizada: diagrama de despliegue

Vista de datos

- Audiencia: los desarrolladores del sistema Nitrate.
- Área: describe cómo la capa de datos estará organizada dentro de una base de datos.
- Documentos asociados: no hay.
- Representación utilizada: diagrama de entidad-relación.

Metas y limitaciones de arquitectura

Es importante reconocer las limitaciones que se habían mencionado en las secciones de dependencias y limitaciones tanto en los documentos de visión y el Project Charter. Lo siguientes requerimientos son claves a la hora de implementar y diseñar la arquitectura (cada categoría se puede encontrar en el documento de requerimientos):

- Requerimientos funcionales: se tomarán como la base para la implementación de la arquitectura.
- Requerimientos de seguridad: se tomará como una prioridad a la hora de diseñar la capa de datos y cómo esta es accesada. Habrán cuentas de usuarios e encriptación de datos.
- Requerimientos de información: se cree que la información en un futuro podría crecer a grandes dimensiones. La implementación y el acceso a los datos deberá ser de forma eficiente.
- Requerimientos de usabilidad: se espera que la aplicación venga a resolver la forma manual de realizar los cálculos con una gran satisfacción al usuario. Este requerimiento va más enfocado al diseño de la interfaz gráfica. No será contemplado en este documento de arquitectura del software.

La arquitectura tendrá las siguientes limitaciones a la hora de implementarse. Se enfocará más a las herramientas a utilizar. Otro tipo de limitaciones como se dijo anteriormente, se mencionó en el documento de visión y Project Charter:

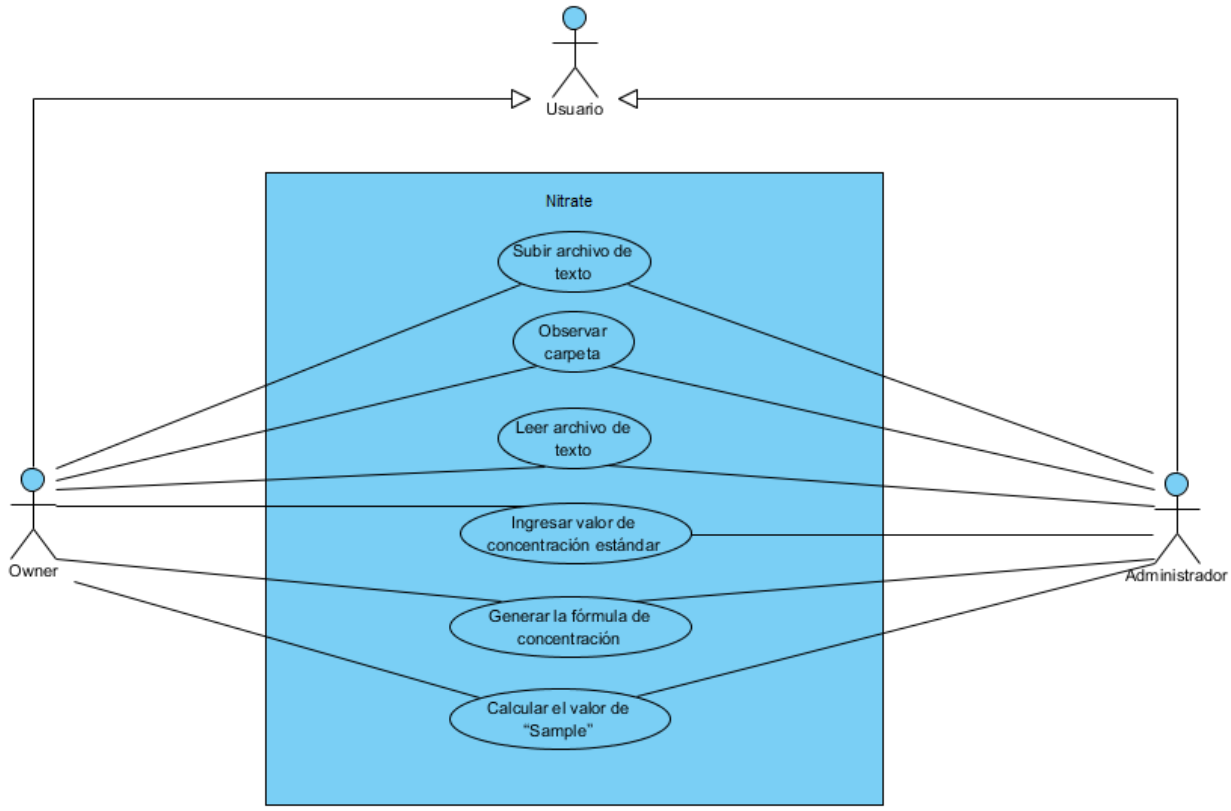
- La aplicación de escritorio se desarrollara con la herramienta Eclipse para la creación de sistemas en Java.
- El manejo al acceso a datos se realizará por medio de un servicio web implementado en nodeJS. Estos se almacenarán en un motor de base de datos MySQL 5.7.
- La aplicación tanto de escritorio como de celular, necesitará conexión a Internet para la validación básica de usuarios.

Vista de casos de uso

Para esta primera versión del documento se tienen los siguientes casos de uso:

Identificador	Caso de uso	Prioridad
UC-001	Subir archivo de texto	Alta
UC-002	Leer archivo de texto	Alta
UC-003	Generar la fórmula de concentración	Alta
UC-004	Observar carpeta	Media
UC-005	Calcular el valor de "*Sample*"	Media
UC-006	Ingresar valor de concentración estándar	Baja

También se obtiene el siguiente diagrama de casos de uso:



Para esta primera iteración se tienen solamente dos tipos de usuario: *Owner* y Administrador. Estos pueden realizar al momento las mismas funcionalidades desde un punto de

vista de casos de uso. La diferencia entre estos radica en que el usuario de tipo *owner* podrá modificar, eliminar y crear otros administradores e inclusives *owners*. La descripción e importancia de estos casos de uso se pueden encontrar con más detalle en el ERS. También el texto de cada caso de uso y los distintos flujos de ejecución se pueden encontrar en el documento de requerimientos; así como los requerimientos funcionales que estos involucran. Sin embargo también se adjuntará en este documento para asegurar su completitud:

UC-001	*Subir archivo de texto*
Versión	1.0
Autor	Josué Arrieta
Dependencias	No hay. Este caso de uso no depende de otro.
Descripción	Un administrador u *Owne*r desea cargar un archivo de texto de su computador a la tabla principal del sistema Nitrate.
Precondición	El archivo de texto deberá haber sido creado anteriormente por fuentes externas al sistema Nitrate.
Postcondición	El archivo de texto es cargado al sistema Nitrate y tal situación es mostrada en pantalla.
Secuencia Normal	1. El usuario selecciona la opción de *Open File* de la pantalla principal. 2. El sistema le muestra al usuario una nueva pantalla con los archivos y directorios del computador. 3. El usuario selecciona el archivo que desea subir. 4. El sistema carga dicho archivo y muestra los datos de él en la tabla principal.
Excepciones	3.1 El archivo seleccionado para cargar es incorrecto, el sistema muestra un error en pantalla de dicha situación. El caso de uso finaliza.
Frecuencia esperada	Se espera que suceda 2 veces por minuto.
Prioridad	Alta
Estado	En construcción
Estabilidad	Alta

UC-002	*Leer archivo de texto*
Versión	1.0
Autor	Josué Arrieta
Dependencias	No hay. Este caso de uso no depende de otro.
Descripción	Un administrador u *Owne*r podrá ingresar una longitud de onda y leer la absorbancia de un archivo (o más) que esté en la tabla principal a partir de una longitud de onda
Precondición	El archivo de texto deberá haber sido cargado anteriormente, ya sea cargado por el usuario o por estar en un directorio observado.
Postcondición	Se muestra en pantalla la absorbancia de dicho archivo de texto para esa longitud de onda.
Secuencia Normal	1. El usuario ingresa en *Wavelength* la longitud de onda deseada. 2. El usuario selecciona la opción de *Absorbance.* 3. El sistema muestra en pantalla la absorbancia de cada archivo en su respectiva fila.
Excepciones	1.1 La longitud de onda ingresada corresponde a un número incorrecto (número de tres dígitos y un decimal), el sistema muestra en pantalla dicho error. El caso de uso finaliza.
Frecuencia esperada	Se espera que suceda 3 veces por minuto.
Prioridad	Alta
Estado	En construcción
Estabilidad	Alta

UC-003	*Generar la fórmula de concentración*
Versión	1.0
Autor	Josué Arrieta
Dependencias	No hay. Este caso de uso no depende de otro.
Descripción	Un administrador u *Owne*r desea crear una calibración con base en la correlación entre las diferentes concentraciones estándares de archivos seleccionadas a partir de una absorbancia dada.
Precondición	El archivo de texto deberá haber sido cargado anteriormente, ya sea cargado por el usuario o por estar en un directorio observado. También los archivos a participar en la calibración deberán tener un valor en la concentración, ya sea ingresada manual o automáticamente.
Postcondición	Se muestra en pantalla la intersección con el eje y, la pendiente y el coeficiente R2 (*Pearson*) de dicha calibración.
Secuencia Normal	1. El usuario selecciona una serie de archivos. 2. El sistema muestra de manera distintiva los archivos seleccionados. 3. El usuario selecciona una columna de absorbancia dada. 4. El sistema muestra de manera distintiva la columna seleccionada. 5. El usuario oprime el botón de *Calibrate.* *6.* El sistema muestra en pantalla la nueva calibración en la tabla de calibraciones, con sus respectivos datos. Además, de los datos de la correlación de *Pearson* obtenidos.
Excepciones	-
Frecuencia esperada	Se espera que suceda 1 vez por minuto.
Prioridad	Alta
Estado	En construcción
Estabilidad	Alta

UC-004	*Observar carpeta*
Versión	1.0
Autor	Josué Arrieta
Dependencias	No hay. Este caso de uso no depende de otro.
Descripción	Un administrador u *Owne*r desea desea seleccionar una carpeta para que el sistema Nitrate la observe y de esta manera cargue automáticamente todo archivo que estará en ella.
Precondición	La carpeta con sus archivos deberán ser creadas anteriormente de manera externa al sistema Nitrate.
Postcondición	La carpeta es observada y todos los archivos en ella cargados
Secuencia Normal	1. El usuario selecciona el submenú de *Tools* de la pantalla principal. 2. El sistema muestra en pantalla las opciones de dicho submenú. 3. El usuario selecciona la opción de *Open Observer*. 4. El sistema muestra en pantalla la carpeta actualmente seleccionada. 5. El usuario selecciona la opción *Browse* para seleccionar una carpeta. 6. El sistema muestra en una nueva pantalla los archivos y directorios del computador. 7. El usuario selecciona la carpeta o directorio a observar. 8. El sistema muestra la carpeta seleccionada 9. El usuario presiona el botón *Start* para iniciar el observador 10. El sistema observa la carpeta y carga los nuevos archivos que ingresen.
Excepciones	4.1 Anteriormente no había una carpeta actualmente seleccionada y el sistema muestra en pantalla dicha situación. Se vuelve al punto 5 y el caso de uso continúa.
Frecuencia esperada	Se espera que suceda 1 vez al día.
Prioridad	Media
Estado	En construcción
Estabilidad	Alta

UC-005	**Calcular el valor de "*Sample*"**
Versión	1.0
Autor	Josué Arrieta
Dependencias	No hay. Este caso de uso no depende de otro.
Descripción	Un administrador u *Owne*r desea estimar la concentración de un archivo a partir de una calibración y absorbancia.
Precondición	La calibración seleccionada deberá haber sido creada anteriormente con el sistema Nitrate.
Postcondición	El sistema muestra en pantalla el valor de concentración calculado para todas las filas.
Secuencia Normal	1. El usuario selecciona una calibración anteriormente realizada en la tabla de calibraciones. 2. El sistema muestra tal fila de manera distintiva. 3. El usuario oprime en el botón de *Concentration*. 4. El sistema muestra en pantalla el valor de concentración calculado para todas las filas.
Excepciones	-
Frecuencia esperada	Se espera que suceda 2 veces por minuto.
Prioridad	Media
Estado	En construcción
Estabilidad	Media

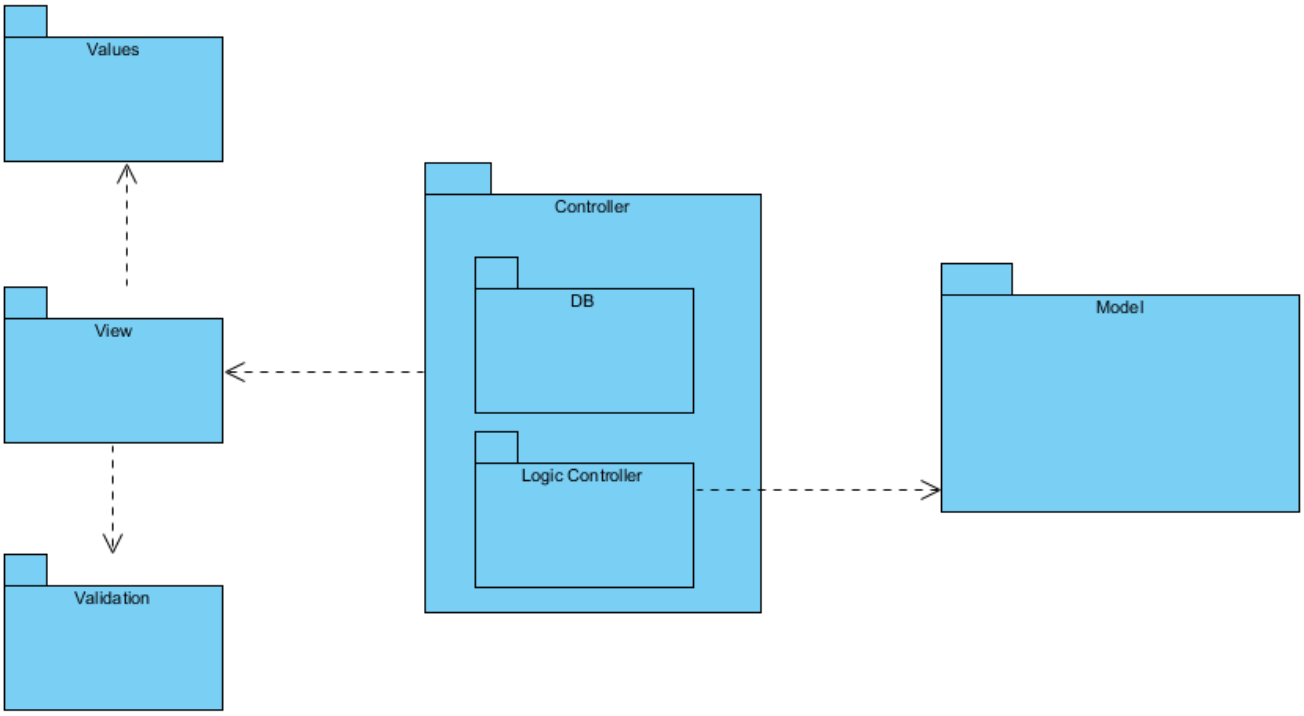
UC-006	*Ingresar valor de concentración estándar*
Versión	1.0
Autor	Josué Arrieta
Dependencias	No hay. Este caso de uso no depende de otro.
Descripción	Un administrador u *Owne*r seleccionar un archivo de tipo STD e ingresar un valor estándar de la concentración de la manera manual.
Precondición	El archivo de texto deberá haber sido cargado anteriormente, ya sea cargado por el usuario o por estar en un directorio observado.
Postcondición	Para los archivos seleccionados se debe mostrar en pantalla la concentración manualmente escogida
Secuencia Normal	1. El usuario selecciona en *Type* el tipo STD para el archivo que desea ingresar la concentración estándar en la tabla principal. 2. El usuario ingresa el valor estándar en la columna *Concentration.* *3.* El sistema muestra en pantalla dicho archivo con tal concentración.
Excepciones	-
Frecuencia esperada	Se espera que suceda 4 veces por minuto.
Prioridad	Baja
Estado	En construcción
Estabilidad	Baja

La prioridad del caso de uso impactará de manera directa a la arquitectura a implementar. Los casos UC-001, UC-002 y UC-003 tendrán la mayor importancia a la hora de implementar la arquitectura y conforme la prioridad vaya disminuyendo así su relevancia.

Vista lógica

Generalidades

Para la arquitectura del sistema Nitrate se tiene el siguiente diagrama de paquetes para esta primera iteración de casos de uso. Se puede notar como se utilizó un modelo de vista y controlador:



Los anteriores son los principales paquetes de la aplicación. Se enlista:

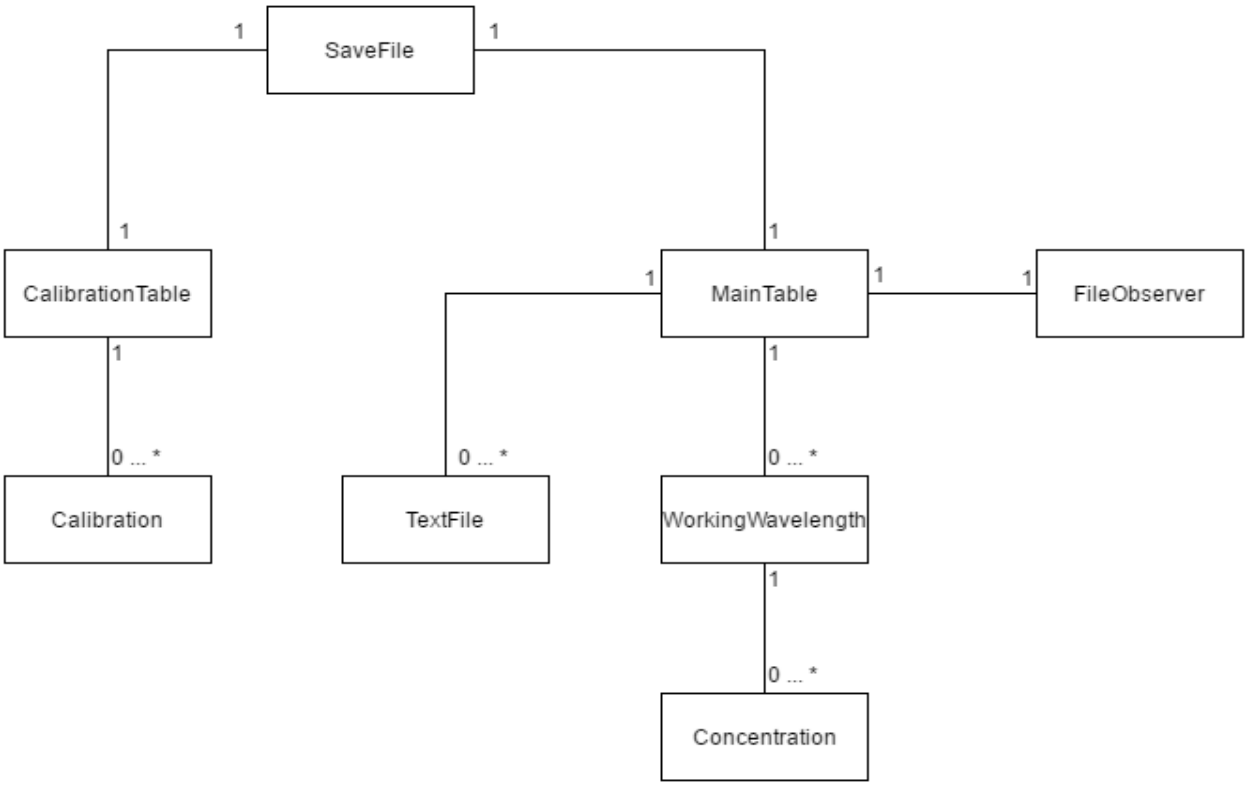
- *Values*: es un paquete con clases de constantes utilizadas para los mensajes de error y configuraciones especiales de la interfaz gráfica. También el modelo lo utiliza para ciertas constantes.
- *Validation*: es un paquete especializado para la validación de entrada de datos del usuario. Esta validación se hará desde la interfaz gráfica.
- *View*: es un paquete que contiene todas las clases referentes a interfaz gráfica del sistema. Será el medio por donde el usuario interactuará con el sistema y podrá ingresar los distintos archivos de entrada.
- *Controller*: es un paquete que será la unión o el medio en que la interfaz podrá interactuar con la interfaz. De esta manera para mejorar la reutilización de paquetes y que sean independientes. Dentro de este paquete hay dos sub-paquetes: 1) DB: se encarga de controlar el acceso a la base de datos y se conectará al servidor web por medio del protocolo http. 2) LogicController este paquete será el controlador que accesará a la lógica de negocios.
- *Model*: este paquete contendrá toda la lógica de negocios. Realizará todo tipo de cálculos y estimaciones.

Paquetes significativos del diseño estructural

En esta subsección se hablará de los paquetes solamente significativos y además las clases que estos poseen que se consideran importantes. Es importante que no se va a hacer un análisis muy profundo al sistema desde un punto de vista lógico, ya que el objetivo de esta sección es solo mostrar una "vista" al modelo.

Model

Dentro del modelo se pueden enlistar las siguientes clases en el siguiente diagrama de dominio:



Se mencionan las clases más importantes:

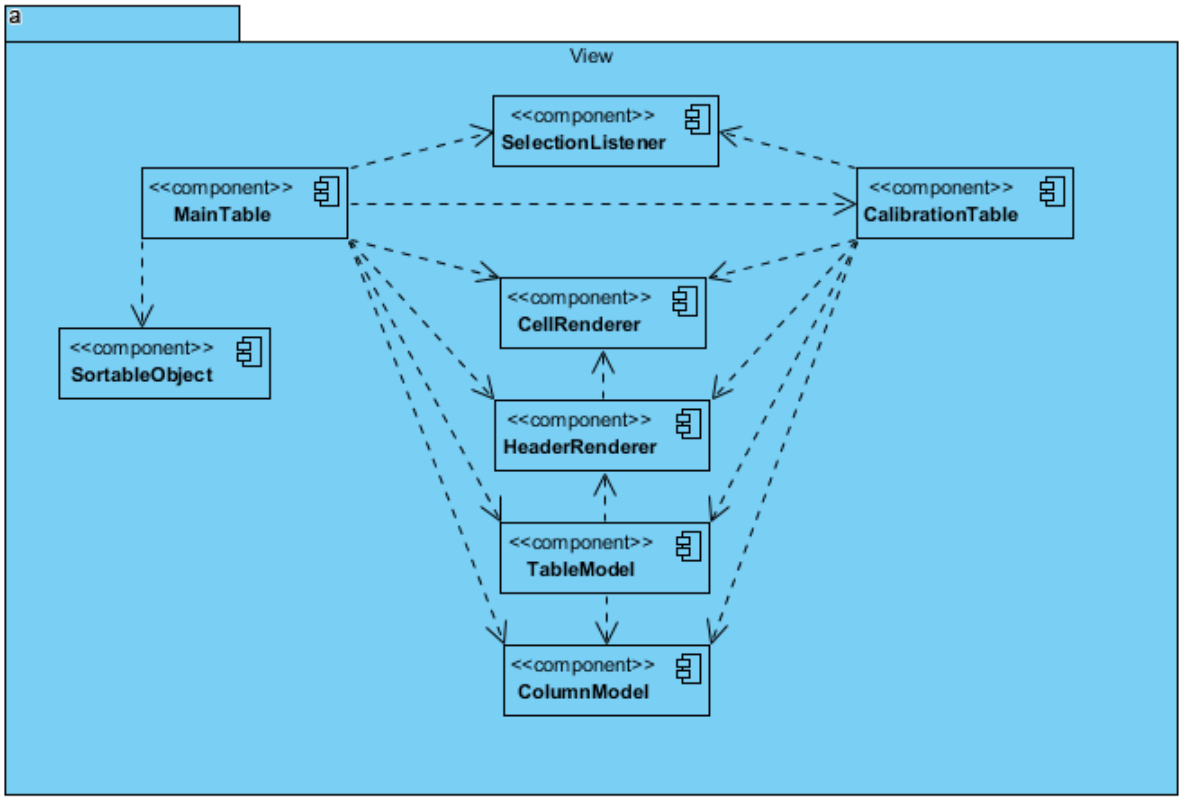
- *MainTable*: se considera la clase más importantes de todas, y es una representación lógica de la tabla principal de la interfaz gráfica. Tendrá que siempre estar igual a lo que se ve en pantall En este estarán todos los archivos, las longitudes de onda para todos los archivos y la concentraciones calculadas a partir de esa longitud de onda. En la segunda iteración se tendrá que poder guardar el estado del proyecto, por lo cual guardar toda esta

información en esta clase es imprescindible. Esta clase puede crear, modificar y eliminar tanto: absorbancias, archivos de texto y concentraciones.

- *CalibrationTable*: tiene el mismo funcionamiento que *MainTable*, excepto que será para la tabla de calibración. Deberá tener el mismo contenido que la interfaz gráfica ya que con esta se podrá cargar el estado de un proyecto. Esta clase podrá modificar, eliminar y crear nuevas calibraciones.
- *FileObserver:* esta clase es un thread encargado de observar la carpeta seleccionada de manera que todo archivo que ingrese a esta carpeta es cargado a *MainTable*. En la sección de: Vista de Procesos se hablará con más detalle acerca de esta clase.

View

Se tiene un diagrama de componentes de las clases consideradas como importantes:



Se quieren rescatar las siguientes dos clases:

- *MainTable:* es la tabla que el usuario ve y siempre deberá estar igual al modelo con su clase respectiva. Esta deberá mostrar (y tener toda interfaz para realizar) archivos, concentraciones y absorbancias. También hay un gran número de detalles de usabilidad

que esta posee tales como: copiado y pegado de objetos, resaltamiento de columnas, orden de archivos cronológico, cambios de tipos automáticos, alargamiento si se desea de columnas, las concentraciones son insertadas a la par de su absorbancia, etc. Posee alto nivel de complejidad y se considera que la gran complejidad del proyecto está en esta clase y cómo representarla y actualizarla en el modelo lógico.

- *CalibrationTable*: es la tabla de calibración que el usuario verá en pantalla y deberá estar igual al modelo lógico con su clase respectiva. Deberá mostrar (y tener toda interfaz para realizar) calibraciones as partir de absorbancias dada. También posee alto número de características de usabilidad tales como: cuando se selecciona una fila se muestran resaltadas las absorbancias con la que se forma, copiado y pegado de objetos, etc.

Realizaciones de casos de uso

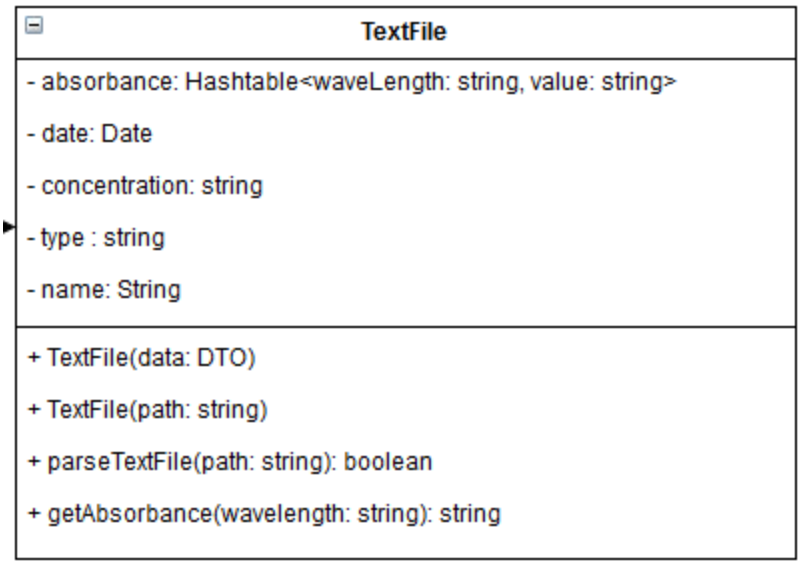
Aquí se menciona cómo las clases o modelos del paquete contribuyen a la realización de los primeros seis casos de uso de esta iteración. Siempre desde una vista lógica.

UC-001 Subir archivo de texto

En el paquete lógico existirá una clase llamada *TextFile*. Esta representa una entidad de un archivo de texto, en donde se tiene una tabla hash con sus valores de absorbancia por longitud de onda. También se guardarán los metadatos del archivo.

UC-002 Leer archivo de texto

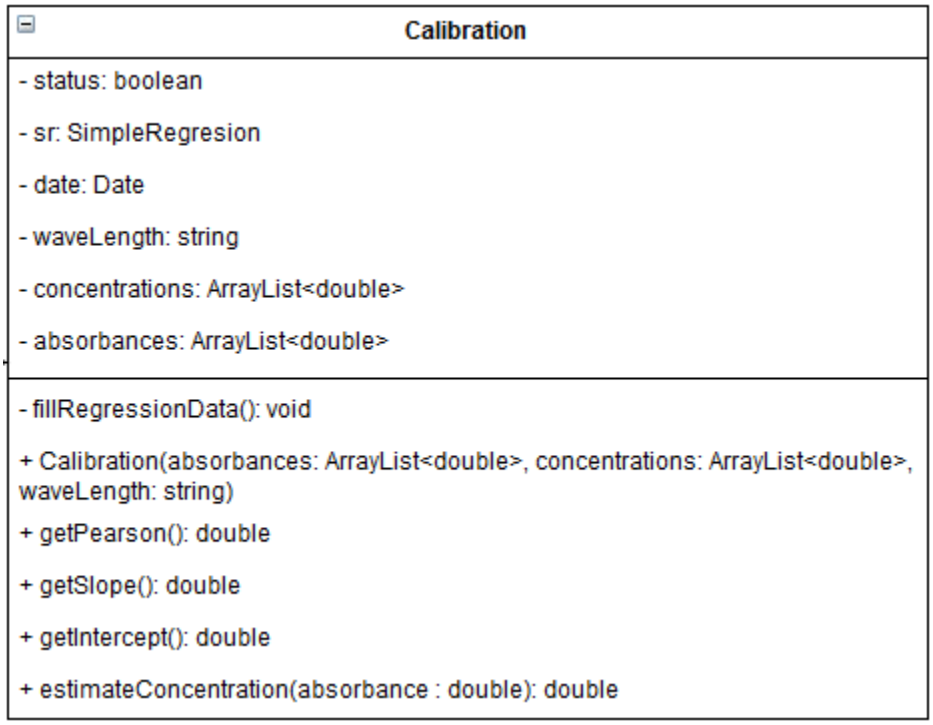
Nuevamente la clase llamada *TextFile* será la encargada de leer el archivo de texto cargado. Es la siguiente:



Con el método *getAbosrbance()* se podrá conseguir una absorbancia a partir de una longitud de onda en el archivo.

UC-003 Generar la fórmula de concentración

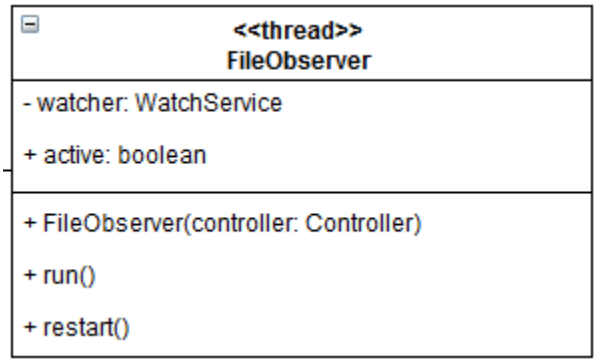
Para realizar este caso de uso se debe generar una calibración. Se tiene la clase *Calibration* para este propósito:



Al crear una calibración se necesita una serie de concentración y una serie de wavelength correspondientes para determinar su correlación (si la hay) entre sí. Luego se pueden conseguir diferentes atributos de la fórmula tales como el *Pearson* (*getPearson*) o la pendiente de la recta (*getSlope*) o la intersección con el eje y (*getIntercept*).

UC-004 Observar carpeta

Como se mencionó anteriormente y se mencionan en la sección de Vista de procesos se tiene la clase *FileObserver*:



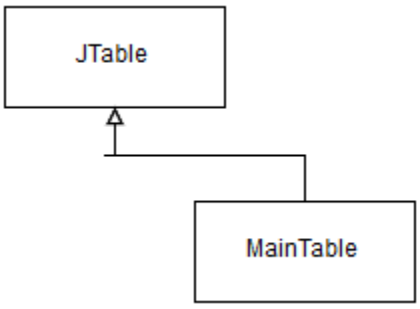
Estará pendiente de un directorio en particular para detectar si un archivo nuevo es ingresado.

UC-005 Calcular el valor de "Sample"

El estimar una concentración se puede lograr a partir de una calibración dada, por lo tanto con el método *estimateConcentration()* en la clase *Calibration* se podrá

UC-006 Ingresar valor de concentración estándar

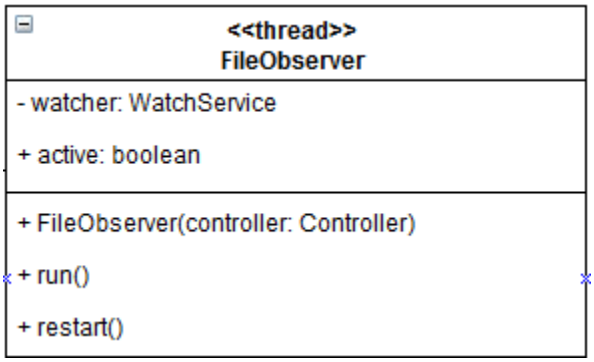
Este caso de uso no requiere interacción con el modelo lógico y será algo que se manejará en la interfaz:



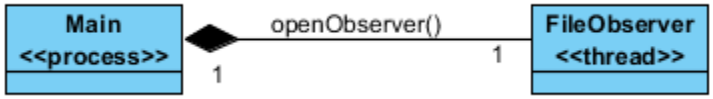
La única celda que el usuario puede ingresar datos es en la de concentración (manual). De modo que *MainTable* debe permitir que se pueda seleccionar un archivo de tipo STD y luego que se pueda editar la columna correspondiente. Esta acción es imprescindible ya que nos permitirá realizar calibraciones más adelante. También se agrega una funcionalidad extra que si se quiere editar el campo de concentración, se pase de tipo automáticamente a STD.

Vista de proceso

Para esta primera versión del sistema, para la realización de los 6 casos de uso en la aplicación de escritorio solo se necesitará un proceso y será el principal. Se le llamará *Main*. Este se encargará de toda la funcionalidad e hilo de ejecución del programa. Además de *Main* se necesitará un *Thread* al que llamaremos *FileObserver*. Este está representado por la siguiente clase del modelo lógico:

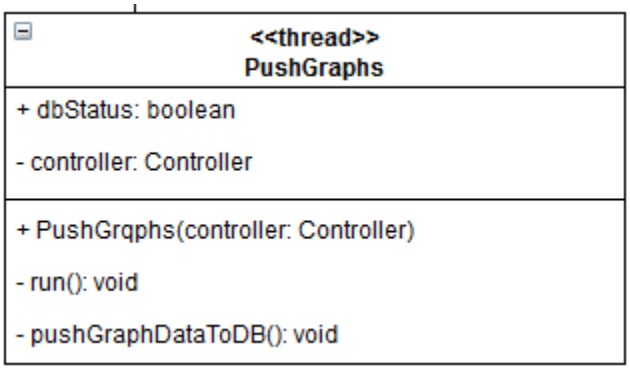


Este es creado por *Main* de la siguiente manera:

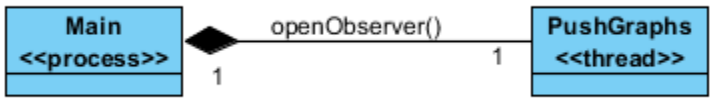


Cada vez que se realice el caso de uso UC-004: Observar carpeta se creará *FileObserver*. Este morirá cuando otra carpeta se ponga a observar o se cierre la carpeta; o bien se cierre la aplicación. El objetivo de *FileObserver* es constantemente revisar un directorio escogido por el usuario, de manera que observe si entra algún archivo de texto a este. De esta manera el *thread* cargará todo archivo añadido de forma automática al sistema Nitrate.

Es importante mencionar que para futuras iteraciones se considera la utilización de un *Thread* más. Se considera pertinente mencionar este *thread* en esta versión del documento por su alta importancia. Tal *thread* está representado en la siguiente clase del modelo lógico y se llamará *PushGraphs*:



Este se creará por *Main* de la siguiente manera:

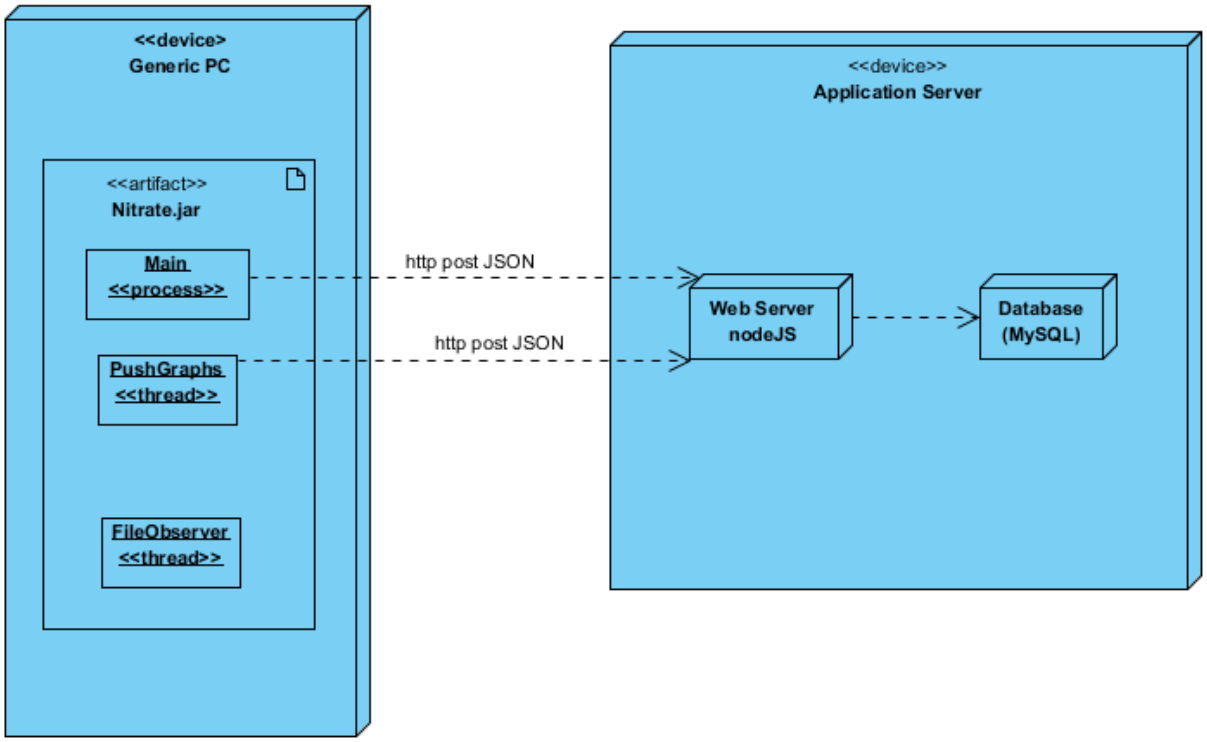


Cada vez que se realice el caso de uso UC-004: Observar carpeta se creará *PushGraphs*. Este morirá cuando otra carpeta se ponga a observar o se cierre la carpeta; o bien se cierre la aplicación. El objetivo de *PushGraphs* es constantemente hacer una llamada al servidor y enviar un JSON con los datos de los gráficos del usuario; para actualizarlos en la base de datos. Esto lo hará con el método *pushGraphDataToDb*(). Si se compara con el comportamiento de *FileObserver* es sumamente similar.

Vista de despliegue

Para esta primera iteración el servicio estará hosteado en un servidor. Solo habrá una configuración del sistema; esta será la Configuración Estándar. Este contendrá el servicio web y el host de la base de datos. Estará almacenado en la nube de Amazon. Es importante mencionar que el computador de escritorio no tiene ninguna característica en general y se considera una PC

genérica. Con el sistema Nitrate esta podrá conectarse con el servidor y poder consultar los distintos usuarios y gráficos que posee. También podrá crear nuevos usuarios y actualizar gráficos (está la realizará como se dijo en la sección anterior *PushGraphs*). *FileObserver* no tendrá ninguna relación directa con el servidor, ya que solo cargará archivos localmente. Esta información acerca de los servicios que se proveen en la base de datos está más detallada en la subsección más abajo de este documento: Vista de datos. Se tiene el siguiente diagrama de despliegue:



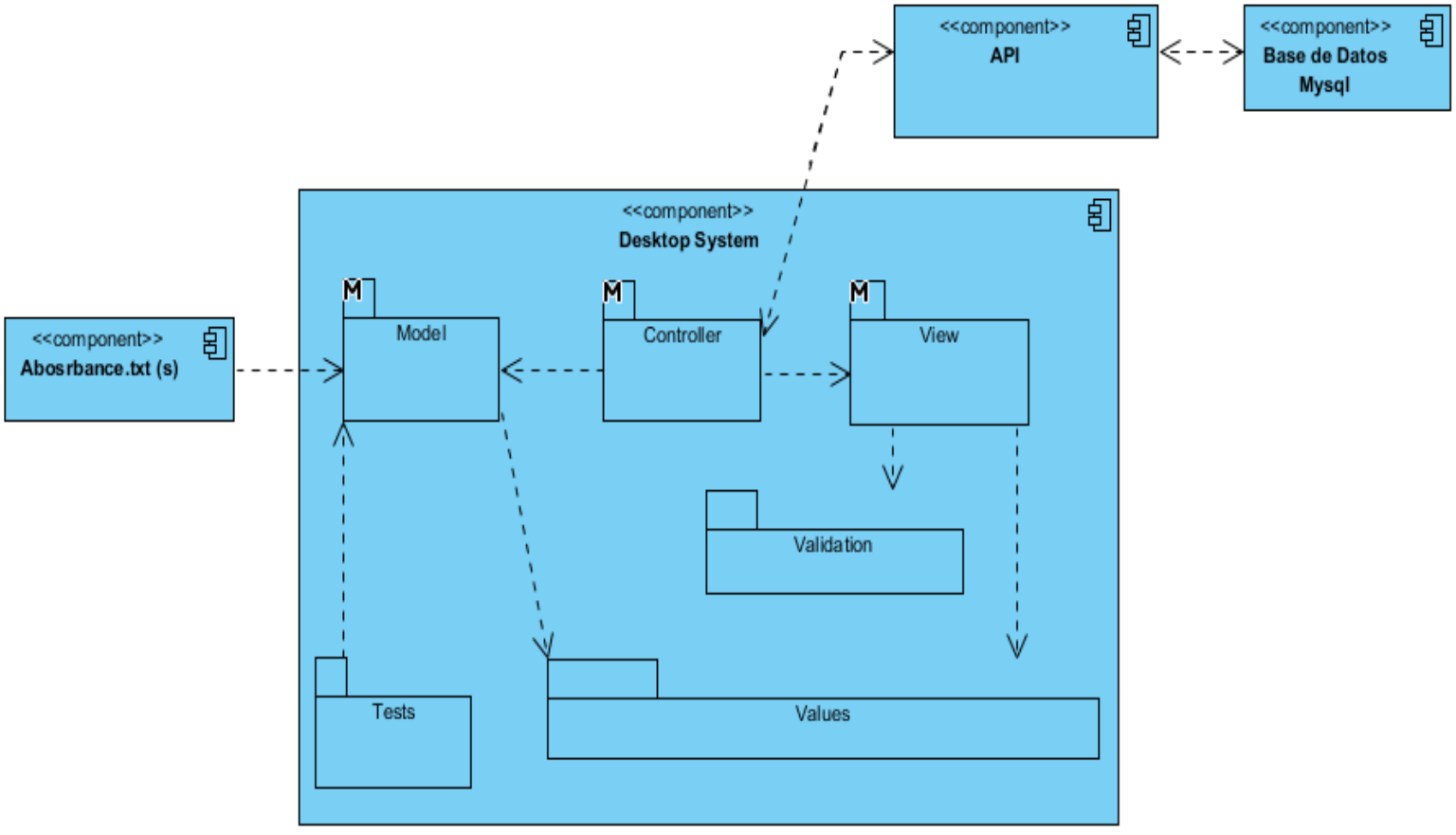
Vista de implementación

Generalidades

Para la implementación se busca utilizar el patrón MVC, por lo tanto se tendrán tres paquetes bien definidos representando el *model*, *controller* y *view*. Donde el *model* sólo podrá tener clases lógicas, sin ningún componente que ligado a una interfaz gráfica. El *controller* tendrá acceso a tanto el *model* como el *view* ya que este servirá de puente entre estos paquetes. El *view* tendrá solo clases que manejen aspectos de la interfaz gráfica, no deberá tener ningún componente del modelo. Este tipo de estructura de tres capas nos permite reutilizar código ya sea del modelo o de la vista en el futuro, ya que son independientes entre sí.

Además se utilizaran diferentes paquetes que representarán las capas transversales que van a manejar diferentes aspectos del sistema. El paquete *Values* contendrá constantes con valores de la interfaz gráfica o datos necesarios para el modelo, este paquete sólo puede tener clases con valores estáticos o enums. Para la validación de datos y entradas se utilizará un paquete *Validation* el cual consta con clases de métodos estáticos que serán llamados para verificar diferentes aspectos de cualquier otra capa. Con el fin de generar pruebas al sistema y automatizarlas, se tendrá un paquete de *test* el cual contendrá clases que pueden ser ejecutadas por JUnit para correr pruebas de manera automática.

Para satisfacer la parte *online* se tienen dos componentes más, la base de datos la cual se encarga de almacenar los usuarios y datos relevantes del sistema y el API el cual sirve de puente entre la aplicación de escritorio y en el futuro la aplicación móvil con la base de datos, de tal manera que estos dos sistemas utilicen los mismos datos con una misma interfaz entre ellos. A continuación se muestra un diagrama de componentes de manera general, las tres capas principales se especificarán más adelante. Las dependencias para la implementación de las capas: *Validation*, *Values* y *Tests*; quedan bastante reflejadas en el siguiente diagrama por lo tanto no se especificara más.

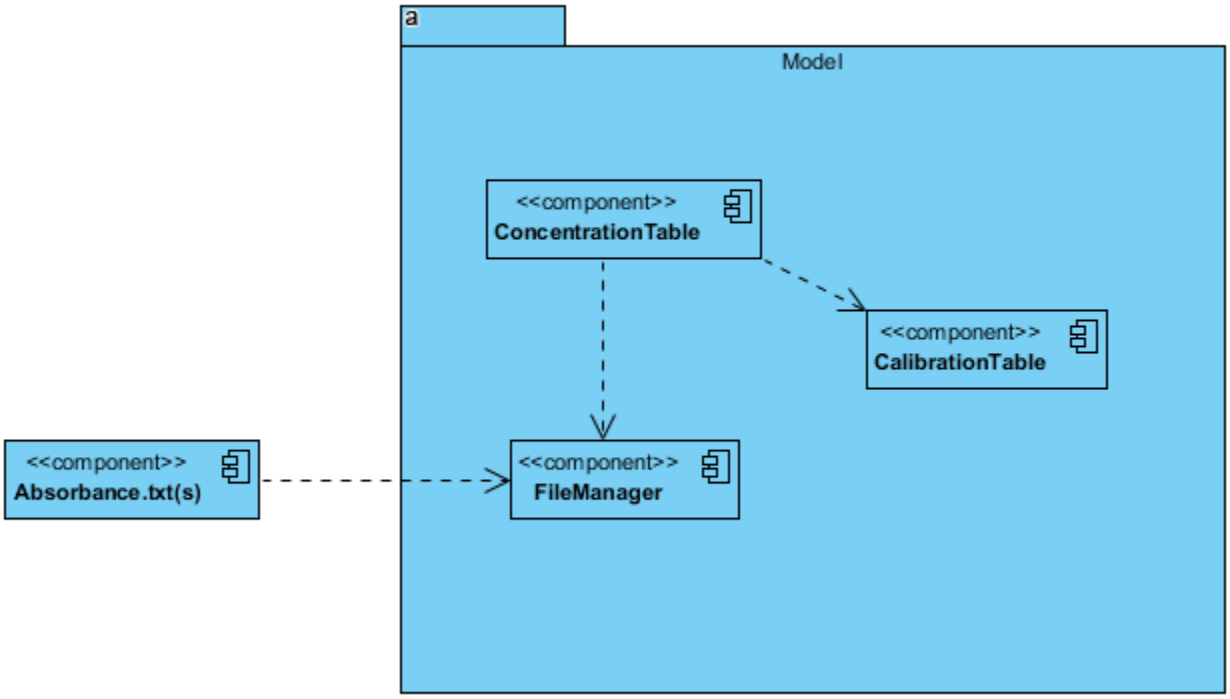


Capas

1. Model

Esta capa contendrá la lógica de negocios del sistema. Se divide en los componentes:

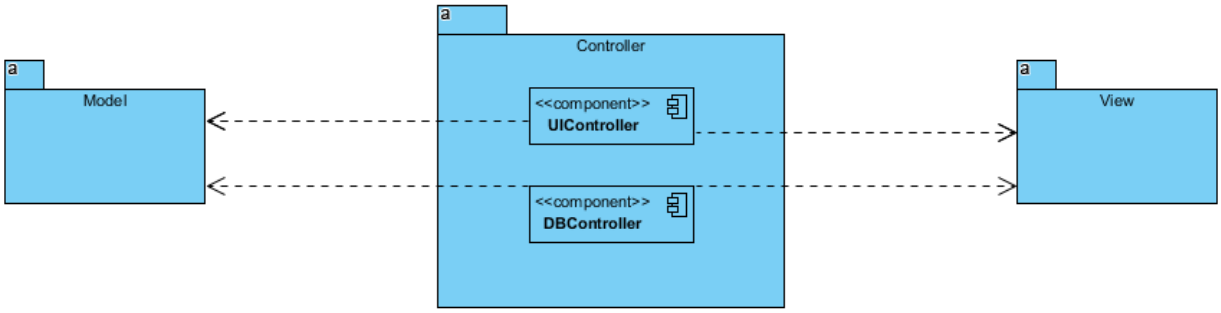
- *ConcentracionTable*: manipula las concentraciones y modifica sus datos
- *CalibrationTable*: manipulas las calibraciones y modifica sus datos
- *FileManager*: procesa los archivos y los guarda



2. Controller

Esta capa contiene los controladores necesarios para manipular los datos de la capa *model* a la capa *view*. Se divide en los componentes:

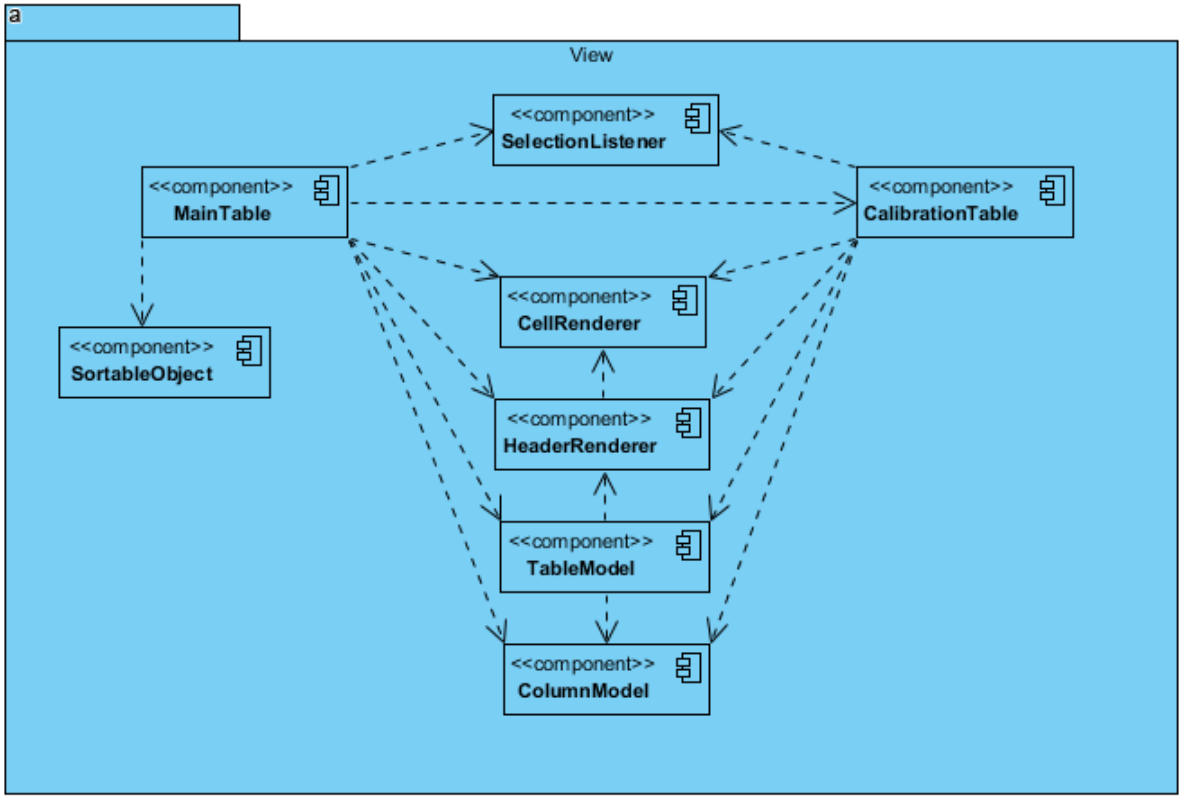
- UIController: se encarga de manipular métodos entre *model* y *view*
- DBController: maneja el control del acceso la base de datos y asigna valores al *model* o *view* según lo requiera.



3. View

Esta capa contiene todo aspecto de la interfaz gráfica, cualquier manejo de datos visual que el usuario debe poder observar debe estar en esta capa. Se incluyen los componentes de software más relevantes para la implementación de la interfaz. Se divide en los componentes:

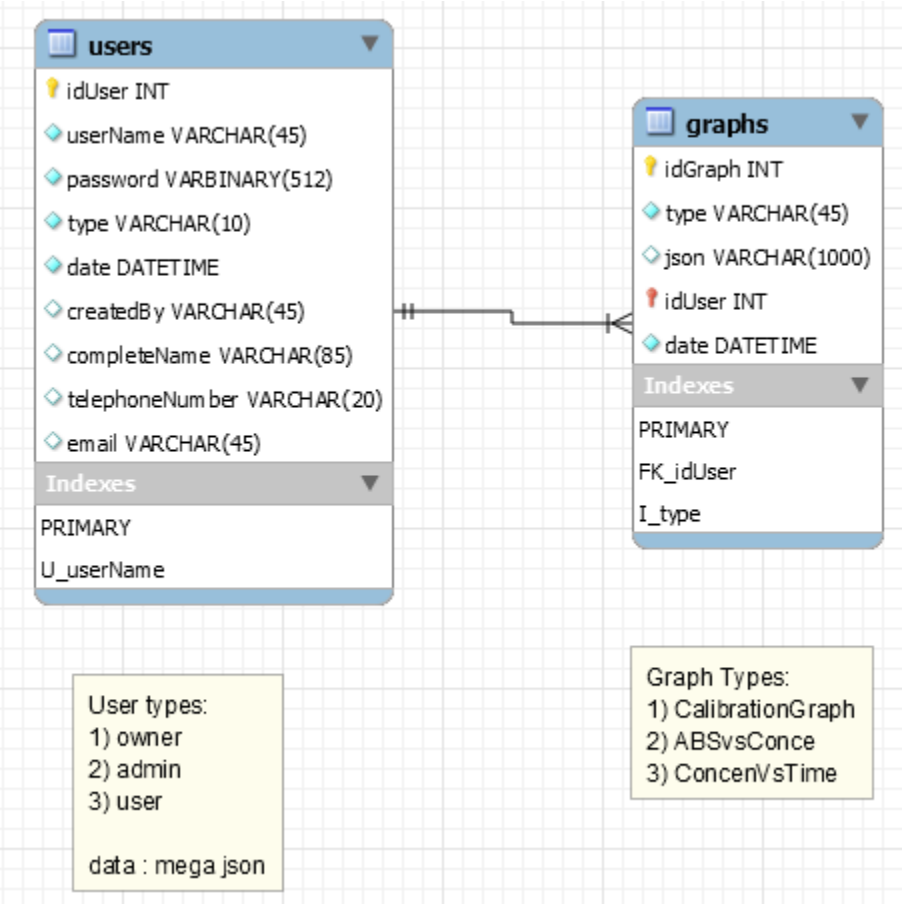
- CalibrationTable
- MainTable
- CellRenderer
- HeaderRenderer
- SortableObject
- ColumnModels
- TableModels
- SelectionListener



Debido a que el proyecto implica mucha manipulación de la parte visual, es necesario recalcar estas dependencias dentro de la capa de vista, ya que afectan de manera significativa el funcionamiento del mismo.

Vista de datos

Para el almacenamiento constante de información se tendrá el siguiente diagrama de entidad relación MySQL:



Debido a que se espera que en un futuro el tamaño de la información puede crecer, se optó por la creación de solo 2 tablas: *user* y *graphs*. Representa a la entidad usuario del sistema Nitrate (hay tres tipos de usuarios) con sus respectivos gráficos (también tres tipos). Esto con el propósito de evitar los *joins* entre tablas ya que son los más costosos. También se pudo haber tenido una tabla especializada para cada usuario, pero se optó por tener una tabla general para aumentar la eficiencia. También se puede observar que para el tipo ya sea de gráficos o de usuarios, no se opta por tenerlos en una tabla separada; se colocan en la misma tabla. Esto con el propósito de nuevamente evitar los *joins* que se tengan que hacer. Se sacrifica levemente la normalización del esquema a cambio de mayor eficiencia.

Para los datos de los gráficos, se optó por un JSON. Este es el tipo de archivo estándar que se utilizará para la aplicación Nitrate y en toda su arquitectura.

También se puede observar la utilización de índices. Se coloca un en el tipo de gráfico, ya que se espera que esta consulta se realice constantemente: buscar un gráfico para un usuario de acuerdo a su tipo.

Para el acceso a estos datos es importante mencionar que no se crearon ni *triggers* ni vistas. Pero si los siguientes procedimientos almacenados:

Para efectos de seguridad todo procedimiento almacenado debe recibir a un usuario válido para poder ejecutarse; de la otra manera se levanta una excepción. Se tienen los siguientes:

- *deleteUserByUsername:* elimina a un usuario dado con sus gráficos. Si se elimina un administrador también sus usuarios creados con sus respectivos gráficos. Solo los administradores y *owner* pueden ejecutar este procedimiento.
- *insertUser:* inserta un usuario. Solo administrador y *owner* pueden ejecutar este procedimiento. Si se inserta un usuario con *username* repetido ocurrirá un error. También se insertan sus tres tipos de gráficos respectivos.
- *validateUser:* devuelve el tipo de un usuario dado, y valida si tal combinación de usuario/contraseña es correcta. Cualquier tipo de usuario lo podrá ejecutar. Devuelve nulo en caso de que no sea correcta.
- *selectUserByUsername:* selecciona la información de un usuario de acuerdo a su username. Solo los administradores y *owner* pueden ejecutar este procedimiento.
- *selectAllAdministrators:* devuelve la información de todos los administradores de un *owner* dado. Solo *owner* podrá ejecutar este procedimiento.
- *selectAllUsers:* devuelve la información de todos los usuarios para un *owner* dado. Solo *owner* podrá ejecutar este procedimiento.
- *selectAllUsersForAdministrator:* selecciona la información de todos los usuarios creados por un administrador u owner dado. Solo estos podrán ejecutarlo.
- *selectUserGraphs:* selecciona los tres tipos de gráfico para un usuario dado. Se puede especificar el tipo si se desea uno en específico. Cualquier tipo de usuario puede ejecutar este procedimiento.
- *updateGraphForUser:* actualiza un tipo de gráfico dado para un usuario específico. Recibe un json para los datos del gráfico. Solo los administradores y *owner* pueden ejecutar este procedimiento.

- *updateUser:* actualiza todos los campos de un usuario dado. Solo los administradores y *owner* pueden ejecutar este procedimiento.

Tamaño y rendimiento

Los distintos requerimientos y necesidades de rendimiento se especificaron en la subsección: Requerimientos de rendimiento. Los requerimientos de rendimiento no son muy predominantes para esta primera versión, sin embargo se tomaron ciertas decisiones de arquitectura para tener altos tiempos de ejecución: tablas *hash* tanto para todos los archivos, donde su llave corresponde su fecha de creación; y también se usó otra tabla hash para el archivo para la correspondencia entre absorbancia y longitud de onda. Esta estructura de datos permite que el tiempo de acceso sea de 1. Se prefiere por encima siempre utilizar un hash en lugar de una lista simple; ya que los valores de las llaves se pueden encontrar en la interfaz gráfica.

Es importante mencionar que para las primeras versiones se espera una cantidad muy baja de usuarios utilizando todo el sistema; de manera que para esta primera versión no se especificará tales requerimientos. Se estima alrededor una cantidad máxima de 5 usuarios consultando la base de datos y el servidor Pero se podría realizar este análisis para futuras versiones de este documento de arquitectura y cómo el tema será manejado.

Se espera que nuevamente no en estas primeras versiones del sistema Nitrate, pero si para futuras versiones que el volumen de datos a manejar podría ser bastante alto. Como se mencionó en la vista lógica para lidiar con esta situación se utilizaron distintos índices y un esquema no normalizado de base de datos, pequeño, pero rápido. Este requerimiento hizo tomar esta decisión de arquitectura.

Calidad

En esta sección se mencionan ciertos atributos de calidad que el sistema debe tener y cómo se acoplará la arquitectura a estos atributos:

- Atributo de extensibilidad: se planea que el sistema se puede actualizar cuando el sistema está corriendo. A pesar de que este equipo no será el encargado del mantenimiento, se planea pensar una arquitectura fácil de actualizar. Esto aplica más a la base de datos, pues la versión de escritorio y de celular se actualizarán simplemente consiguiendo la nueva

versión. Esto se logra creando scripts que en lugar de crear el esquema, más bien lo actualicen.

- Atributo de portabilidad: se espera que el sistema Nitrate se pueda adaptar a cualquier equipo. Por esta razón se utilizó java como lenguaje de programación, ya que sus programas corren en una máquina virtual independiente al sistema donde se esté corriendo. También se menciona que para la aplicación de celular se utiliza una webapp, también con el mismo propósito de que se adapta fácilmente a Android o iOS.
- Atributo de mantenibilidad y reusabilidad: se espera que la arquitectura sea fácil de mantener, modificar y reutilizar. Para lograr este requerimiento el diseño de la lógica de negocios deberá implementar los principios de SOLID. Se hará énfasis al single *responsibility principle*, *open/closed principle* y *liskov substitution principle*. Además se optó por patrones de diseño, especialmente MVC que permite la separación entre la lógica de negocios y la interfaz gráfica para que se pueda reutilizar en un futuro. Este aspecto es uno de los que más marca el diseño de la arquitectura lógica.
- Atributo de testabilidad: se espera que la arquitectura sea fácil de probar. Para este requerimiento, se cuenta con un paquete dedicado al *testing* automático, de manera que realizar pruebas de regresión sea muy eficiente. Se tendrán dos clases: una para pruebas unitarias y otra para pruebas de integración.
- Atributo de privacidad: se espera que la aplicación sea seguro. De esta manera en la vista de datos las contraseñas de los usuarios se encuentran encriptados con SHA-512 bits. También todo JSON enviado será encriptado.

Glosario

Se recomienda consultar el glosario que se encuentra en el ERS para su mayor entendimiento del sistema y de los términos que se utilizan. Se tienen los siguientes términos para este documento:

Término	Definición
Arquitectura	En software, una arquitectura dice cómo el software está construido por dentro: sus componentes y cómo estos se relacionan.
Caso de uso	Es una descripción de los pasos o las actividades que realiza un usuario para satisfacer un objetivo.
Diagrama de clases	Tipo de diagrama que nos muestran como las instancias de una clases se relacionarán (comportamiento) y cómo estas están compuestas (estado).
Diagrama de despliegue	Es un tipo de diagrama que nos dice cómo el software y el hardware estará relacionado. Qué tipo de máquinas correrán cuáles procesos.
Diagrama de entidad-relación	Es un diagrama en que las entidades del mundo real se relacionan y cómo estas están constituidas a través relaciones.
Diagrama de paquetes	Es un diagrama que nos dice la descomposición del sistema en agrupaciones lógicas.
Draw.io	Una herramienta que permite editar en tiempo real y de forma compartida distintos diagramas.
Join	En SQL corresponde a la instrucción u operación que se utiliza para pegar o relacionar dos tablas.
SAD	Corresponde a *system architecture definition*. Es este mismo documento.
MVC	Patrón de diseño Modelo Vista Controlador, que supone la separación de la lógica de negocios y de la interfaz gráfica. Promete sistemas reutilizables.
Procedimiento almacenado	Es un proceso pre-compilado que está en la base de datos.
Thread	Es un flujo de ejecución del procesador alterno. Se ejecuta de forma concurrente.
Triggers	Son flujos de ejecución que se disparan cuando ocurre un evento: inserción, borramiento, y actualización de una tabla.
UML	Es un lenguaje comercial estándar que se utiliza para la creación de

	diagramas de arquitectura del software.
Vista	En una arquitectura, corresponde una forma de verla; ya sea desde un punto de vista lógico, más del lado del usuario o físico.