PRÁCTICA FINAL PROGRAMACIÓN III

jTasks

CURSO 2024/2025

Fecha límite de entrega: 12/12/2024
Facultad de Ciencias - Universidad de Salamanca

Introducción

Una empresa nos plantea el desarrollo de una aplicación en Java y para cualquier plataforma de escritorio (Mac OSX, Linux, Windows, etc.), que permita a los estudiantes organizar y llevar un registro de las tareas que realizan a lo largo del curso académico. La aplicación será inicialmente de consola pero la empresa pretende en el futuro añadir otro tipo de interfaces (como interfaz gráfica o incluso que sea un asistente personal que permita añadir estas tareas por voz). Por este motivo es importante que en el diseño de la aplicación se tomen ciertas decisiones para poder crear capas de abstracción en ciertas partes de la aplicación y que la aplicación tenga una buena arquitectura para poder hacer cambios y ser probada de forma automática en el futuro.

Requisitos

Los requisitos funcionales de la aplicación se resumen en las siguientes secciones:

- **Capacidad de realizar operaciones CRUD de las tareas**: las tareas tendrán cierta información obligatoria que se detalla en el apartado de diseño. La interfaz gráfica deberá permitir las siguientes operaciones CRUD:
    - Dar de alta una tarea a partir de los datos proporcionados por el usuario
    - Listados:
        - Listado de todas las tareas que están sin completar, ordenadas por prioridad (mayor a menor).
        - Listado del historial completo de tareas (completadas o no).
    - Detalle de la tarea. Solo se mostrará el detalle de la tarea seleccionada de uno de los listados anteriores y desde ahí se dará la opción al usuario de:
        - Marcar como completada/pendiente (se mostrará una opción distinta dependiendo del estado).
        - Modificación de toda de la información salvo el código de identificación que será único.
        - Eliminar la tarea.

- **Exportación/Importación como CSV**: la aplicación permitirá exportar/importar en formato CSV las tareas dadas de alta ordenadas por fecha (más recientes primero). La política de importación será la de no importar aquellas que tengan el mismo identificador único.

- **Exportación/Importación como JSON**: Similar al anterior pero en formato JSON.

- **Persistencia**: el programa, **deberá ser capaz de recuperar el estado de la aplicación con las tareas disponibles**. Para este fin, como opción por defecto, se utilizará el mecanismo de serialización en Java para guardar la información del modelo. Se cargará la información en el inicio de la aplicación y se guardará al salir de esta. Esta persistencia se abstraerá con una interfaz que permitirá cambiar la persistencia de la aplicación en su inicio mediante un parámetro por

consola. Adicionalmente esta persistencia podrá cambiar realizándose en un proveedor externo, en este caso Notion.

- **Interfaces de usuario**: el prototipo deberá contar con una interfaz por consola con un menú interactivo pero también estará abierto a extensión ya que en futuro se quieren implementar otras interfaces (GUI, voz, etc). Un buen diseño de POO es vital en esta parte. Se deberán implementar al menos la primera de las siguientes interfaces de usuario:

    - Interfaz de consola simple interactiva: Dispondrá de un menú y submenús para distintas opciones. Será la opción por defecto de la aplicación.
    - CLIView: en el futuro la empresa quiere añadir una opción de usar la aplicación solo por parámetros de consola y en un solo uso.
      [Opcional no evaluable]
    - VoiceView: en el futuro se desea hacer una vista que narra las opciones del menú con un TextToSpeech.
      [Opcional no evaluable]

- **Aplicación robusta y resistente a fallos**: la funcionalidad del prototipo debe de haber sido probada manualmente en modo interactivo, revisando toda la funcionalidad implementada. **Si una característica no funciona la empresa no la valorará.** Se deberán manejar excepciones e informar al usuario de posibles errores debidamente, sin la parada abrupta de la aplicación.

- **Arquitectura MVC**: un punto clave del desarrollo de la aplicación es que siga las guías de diseño del estilo arquitectónico Modelo Vista Controlador tratadas a lo largo del curso en la asignatura. No solo debe funcionar sino que se debe seguir la arquitectura especificada.

- **One More Thing**: el CEO de la empresa es un apasionado de la aplicación Notion y nos exige que sea posible que la persistencia de la aplicación se integre con Notion. El CEO quiere presentar esta característica como extra en la presentación del prototipo, solo así considerará que el trabajo es digno de un 10.

GUÍA DE DISEÑO E IMPLEMENTACIÓN

A continuación se proporcionan una serie de guías de diseño para el desarrollo de la aplicación y que se deberán tener en cuenta.

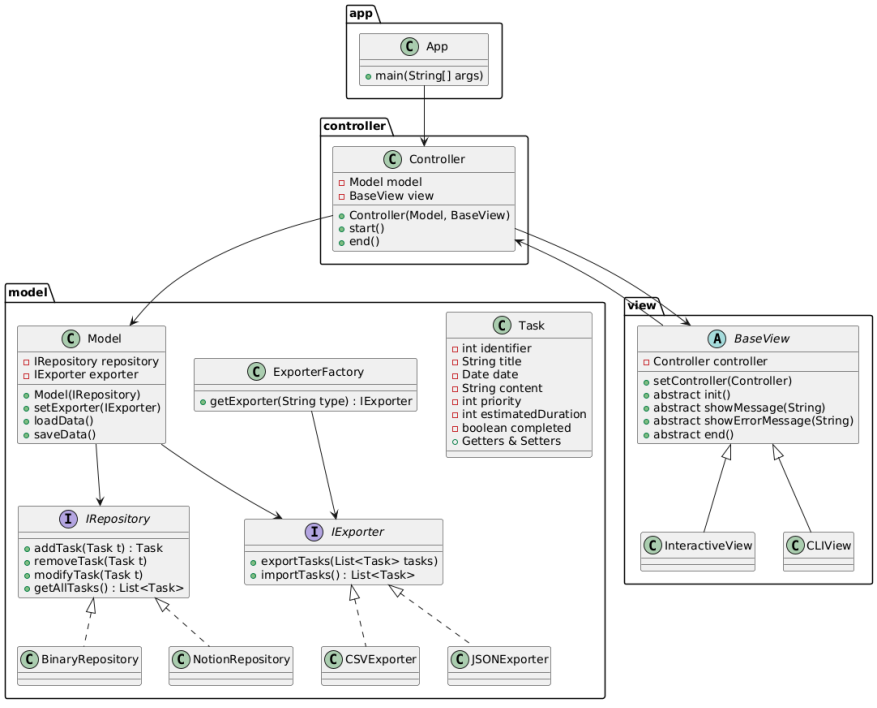
Si no comprende algo o necesita alguna aclaración, escríbalo por el foro de dudas de Studium habilitado para tal fin, resolverá su duda y la de sus compañeros.

Arquitectura MVC:

La aplicación deberá presentar una arquitectura MVC. Esta se deberá armar en el inicio de la aplicación estableciendo los objetos que tomarán parte en la aplicación. **La aplicación siempre trabajará con las clases Base*View, Model* y *Controller***. La instancia de las clases concretas será lo que cambiará dependiendo de los parámetros pasados por consola en String[] args. Para armar el MVC se podrá utilizar el método main de la clase App.java.

- ☐ **Revise previamente los ejemplos de MVC con herencia e interfaces proporcionados durante las clases prácticas.**

A continuación, se presenta un diagrama de clases UML (**incompleto**) de un posible enfoque del ejercicio:



En el esquema no se muestran todas las clases que podría tener el proyecto (esto se lo dejamos a usted), pero sí una guía de cuáles podrían ser las principales. Nótese que el controlador tiene una referencia a *BaseView* (que es una clase abstracta) y que la clase Model tiene atributos de tipos que son Interfaces. Esto significa que el controlador solo

emplea los métodos de dicha superclase *BaseView* y el modelo sólo podrá hacer uso de los métodos de las interfaces *IRepository e IExporter, no conocerá las instancias subyacentes*. Esto abstrae dichas partes de la aplicación y hace que se puedan sustituir fácilmente. Las instancias concretas de **dichos objetos serán configuradas en la inicialización del programa en el *main***, consiguiendo que la lógica de negocio de la aplicación se mantenga intacta, a pesar de cambiar dichos objetos.

Paquete view:

BaseView: es una clase abstracta que tiene los siguientes métodos abstractos y atributos:

Tendrá un atributo:

- `Controller controller.`

Tendrá como mínimo los siguientes métodos abstractos:

- `void init(): inicia la vista y desencadena la lógica de la vista.`
- `void showMessage(): permite notificar de un mensaje al usuario.`
- `void showErrorMessage(): permite notificar de un mensaje de error al usuario.`
- `void end(): finaliza la vista ordenadamente.`

La funcionalidad de los métodos abstractos la implementaran las clases que heredan de **BaseView**, en este caso como mínimo se debe implementar la siguiente:

- **Interfaz de consola simple interactiva:** será una clase que hereda de Base*View* y que implementa los métodos anteriores, elegid el nombre que prefiráis. Será la encargada de mostrar los mensajes de inicio de la aplicación y de iniciar el menú con las opciones del programa al usuario. Gestiona toda la interacción con el usuario y las opciones de la aplicación que son:
  - **Menú CRUD**: se deberán implementar las siguientes operaciones CRUD.
    - **Alta** a partir de los datos proporcionados por el usuario.
    - **Listados**:
      - Listado de todas las tareas ordenadas por prioridad (mayor a menor) y que están sin completar.
      - Listado del historial completo de tareas (completadas o no).
    - **Detalle de alguna de las tareas**. Se mostrará el detalle de la tarea seleccionada de alguno de los listados anteriores y desde ahí se dará la opción al usuario de:
      - Marcar como completa/incompleta (se mostrará una opción distinta dependiendo del estado)
      - Modificación del resto de la información salvo el código de identificación.
      - Eliminar la tarea.
      - Volver al menú principal.
  - **Menú exportación/importación**: se deberá poder exportar todas las tareas a un fichero en el home del usuario en formato JSON o CSV (output.json, output.csv). Del mismo modo se deberá poder importar a partir de esos mismos ficheros. La política de importación será la de no importar aquellas que tengan el mismo identificador único que uno ya existente en el listado de tareas de la aplicación.

Paquete controller:

El controlador será un mero intermediario y orquestador en esta aplicación. Se ocupará principalmente de:

- La inicialización de la aplicación indicando al modelo que cargue la información previa (si existiera) y notificando a la vista que muestre un mensaje con el resultado de dicha operación.
- Indicar a la vista que se inicie.
- Recibir órdenes desde la vista y trasladarlas al modelo, devolviendo los retornos correspondientes o **propagando las excepciones correspondientes**.
  Sí, deberemos utilizar excepciones y propagarlas hasta la vista si fuera necesario. Ya en la vista hacer el try catch e informar al usuario del error concreto dependiendo de la excepción.
- Finalizar ordenadamente la aplicación indicando al modelo que guarde el estado de la misma y finalmente indicando a la vista que muestre un mensaje si todo ha ido bien o se ha producido algún error.

Paquete model:

Existirá una clase que se corresponde con el **modelo** y que debe tener, al menos, los siguientes atributos:

IRepository: se trata de un objeto que implementa la interfaz IRepository que permitirá hacer las operaciones CRUD sobre la colección de tareas interna que posea. Esta colección la podrá haber cargado de un archivo binario o de otra fuente, dependiendo del IRepository concreto:

- En el caso de ser un BinaryRepository se deberá crear una colección de objetos de un fichero binario en el home del usuario llamado tasks.bin si este existiera. Los mantendrá en memoria durante la ejecución de la aplicación y cuando termine la aplicación se guardarán de nuevo los objetos en el mismo fichero binario.
- En el caso de ser un NotionRepository se gestionará el acceso mediante la biblioteca proporcionada. En este caso no se mantendrán en memoria los objetos y se obtendrán siempre a través de la biblioteca, para evitar problemas de sincronización y caché.

Los métodos que implementará esta interfaz deberán ser:

- Task addTask(Task t) throws RepositoryException
- void removeTask(Task t) throws RepositoryException
- void modifyTask(Task t) throws RepositoryException
- ArrayList<Task> getAllTask() throws RepositoryException

Todas lanzan una Excepción personalizada denominada **RepositoryException** que envuelve las posibles excepciones que se puedan producir. Revisar el tema dedicado a las excepciones.

El IRepository concreto los obtendrá por constructor el modelo en el inicio de la

aplicación.

IExporter: se trata de un objeto que implementa la interfaz IExporter y que permite exportar o importar las tareas. Contará con dos métodos export e import para exportar/importar las tareas. Si se produce un error se deberá lanzar una ExporterException personalizada que envuelve las posibles Excepciones que se den. La creación del Exporter concreto se delegará en una clase llamada ExporterFactory que tiene un método estático que recibe como parámetro un string con el tipo de IExporter que se desea construir "csv" o "json" y devolverá una instancia de un IExporter concreto para esos tipos de exportación/importación. El controlador será el encargado de establecer dicho IExporter en el modelo y realizar la exportación posteriormente: (1) obtener el exporter adecuado, (2) establecerlo en el modelo y (3) realizar la exportación/importación.

Clases del modelo de dominio:

Una clase POJO que represente una **Task (tarea)**:

- **identifier**: este deberá ser un valor numérico único. Se deberá controlar que no existan dos tareas con el mismo identificador.
- **taskTitle**: Alfanumérico
- **date**: objeto Date de Java
- **content**: Alfanumérico
- **priority**: entero entre 1 y 5 siendo 5 la mayor prioridad.
- **estimatedDuration**: en minutos: Numérico.
- **completed**: boolean

Obligatoriamente se deberán utilizar los mismos nombres para los campos identifier y content de cara a las pruebas.

NOTAS IMPORTANTES SOBRE EL CICLO DE VIDA DE LA APLICACIÓN

Ejecución de la aplicación y configuración de MVC:

Al ejecutar la aplicación por consola, el usuario podrá pasarle parámetros que definirán cómo se ejecutará la aplicación. Como sabe, los parámetros pasados se encontrarán en String[] args del método main. Cuando es necesario pasar un argumento que tiene espacios es posible utilizar las comillas dobles. La aplicación podrá recibir los siguientes parámetros:

--repository: cuyas opciones serán **bin** o **notion**, siendo **bin** la opción por defecto. Esto hará que el objeto IRepository cargado en el modelo sea o bien un BinaryRepository o un NotionRepository. En el caso de **notion** se le proporcionará la API_KEY y el DATABASE_ID de la tabla como parámetros. El repositorio creado se le pasará por constructor al objeto Model.

- `java -jar app.jar --repository bin`
- `java -jar app.jar --repository notion "API_KEY" "DATABASE_ID"`

Este parámetro define cómo se construye el modelo, por lo que será lo primero que se deberá comprobar.

ONE MORE THING

Notion es una herramienta de productividad muy flexible que permite definir lo que ellos denominan “Databases” y “Pages”. Estas “Databases” permiten almacenar información y presentarla posteriormente de formas muy diversas (con diferentes vistas como tablas, calendarios, galería, etc.). Su flexibilidad hace que pueda ser utilizada para multitud de propósitos (gestión de tareas, calendario, gestión de equipos, automatizaciones, etc.) y la ha convertido en una empresa unicornio con una valoración de 10 Billion $ en pocos años. El CEO de la empresa que nos encarga la aplicación **jTasks** quiere integrar estas Databases con la API que ofrece Notion utilizando una biblioteca disponible para ello. Esta feature la presentará en su keynote en el apartado de “One more thing…”.

El CEO insiste que ha detectado en un estudio de mercado que hay una necesidad que cubrir en la organización de tareas de los estudiantes y otras facetas de su día a día:

“PROBLEMA: La falta de organización de los estudiantes suele ser un problema para que alcancen sus objetivos y esto puede ocasionar estrés, mala organización u olvido de las tareas pendientes.

PROBLEMA: Organización de tareas y seguimiento de hábitos de los estudiantes así como la organización del curso en general, exámenes, contacto con profesores...

PROBLEMA: LA GESTIÓN DEL TIEMPO Y EL ENFOQUE EN LAS TAREAS ES UN PROBLEMA COMÚN PARA LAS PERSONAS QUE TIENEN MÚLTIPLES RESPONSABILIDADES. LA PROCRASTINACIÓN Y LA FALTA DE SEGUIMIENTO PUEDEN HACER QUE NO SE CUMPLAN PLAZOS IMPORTANTES

PROBLEMA: Un Gestor de Tareas SOLUCIÓN: Quizás una clase llamada Tarea con los distintos parámetros de esta así como la fecha o asignatura y el estado en el que se encuentra, "sin empezar", "en progreso" o "completada"

PROBLEMA: APLICACION QUE ORGANICE UN CALENDARIO DE TODAS LAS ACTIVIDADES QUE TE APETEZCA AÑADIR

PROBLEMA: PARA MUCHAS PERSONAS ES DIFÍCIL ORGANIZARSE Y GESTIONAR SU TIEMPO. SOLUCIÓN: APLICACIÓN QUE PERMITA ORGANIZAR TODAS LAS TAREAS Y ACTIVIDADES QUE NECESITEMOS HACER, Y QUE ADEMÁS FUNCIONE CON INTELIGENCIA ARTIFICIAL.

PROBLEMA: ORGANIZAR UN HORARIO EFICIENTE DE ESTUDIO PARA UN ESTUDIANTE UNIVERSITARIO.

*PROBLEMA: GESTIÓN DE TAREAS Y HÁBITOS SOLUCIÓN: LA TEMÁTICA ESTARÍA CENTRADA EN AYUDAR A LAS PERSONAS A ORGANIZAR SUS ACTIVIDADES DIARIAS, ESTABLECER METAS Y REALIZAR UN SEGUIMIENTO DE SUS HÁBITOS. ALGO ASÍ COMO **NOTION**.”*

Si investiga en profundidad la herramienta Notion podrá comprobar que su flexibilidad permite convertirla con poco esfuerzo en una aplicación que da respuesta a estas necesidades.

Notion SDK

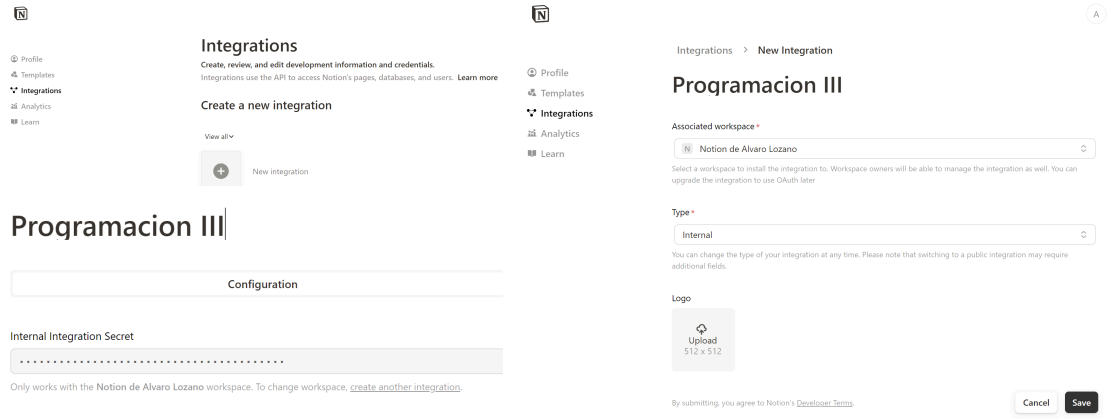
En esta sección se detalla cómo implementar la clase **NotionRepository** que implementa la interfaz IRepository presente en el modelo. Esto permitirá que en lugar de almacenar la información en un fichero binario con serialización, la aplicación realice la misma funcionalidad pero almacenando la información en "Databases" de Notion.

Se proporciona un JAR de la biblioteca para realizar operaciones CRUD y un proyecto de ejemplo que debéis tomar de referencia:

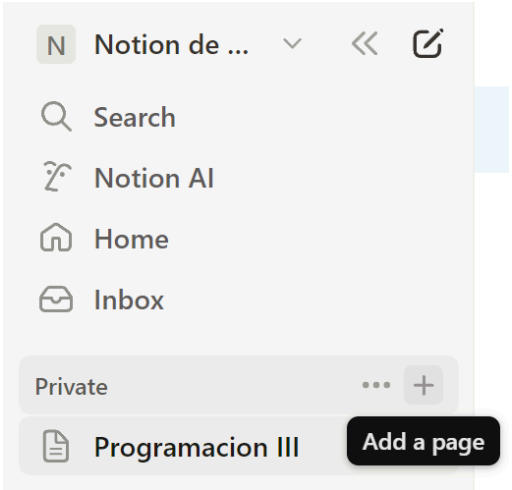
PROYECTO DE REFERENCIA

Como pasos previos a utilizar la biblioteca será necesario:

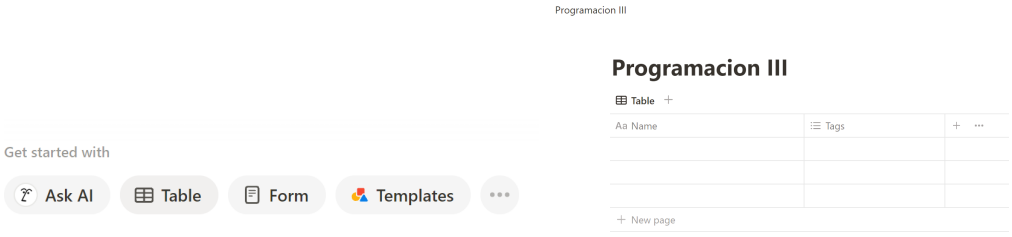
1. Crear una cuenta en Notion.
2. Crear una integración con un nombre determinado (Programación III) y obtener una API KEY (Internal Integration Secret) en la siguiente página : https://www.notion.so/profile/integrations



3. Ahora será necesario crear una nueva Page en el menú lateral. Se le ha dado el nombre de Programación III.

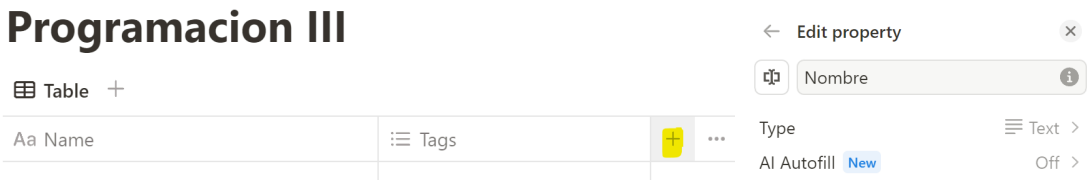


4. Se añadirá una vista de Tabla en la parte inferior de la página:



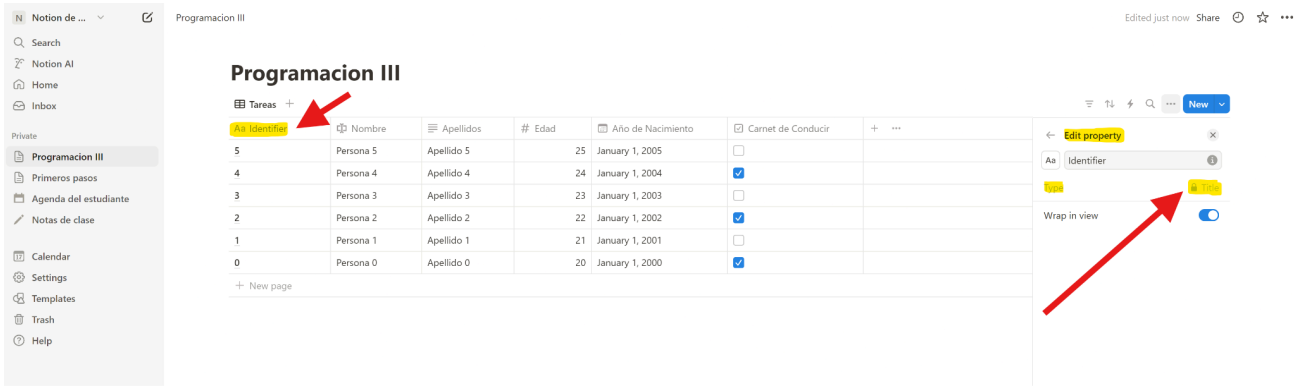
Ahora se deberán añadir las propiedades que coincidan con los atributos de nuestro modelo Tarea. Para añadir estas propiedades (columnas de la tabla) basta con hacer click en el símbolo + y añadir columnas dándole un **nombre** y un **type concreto (Text, Checkbox, Number, etc.). Estos dos aspectos son importantes ya que definen qué se puede almacenar en dicha columna.**

Por otro lado, **la primera columna de la tabla tiene un tipo especial fijo denominado Title que no podemos cambiar**. Será este primer campo el que se utilizará como identificador único para nuestros registros en lugar del Id interno que tienen todos los registros de las Databases de Notion (llamados Pages también).

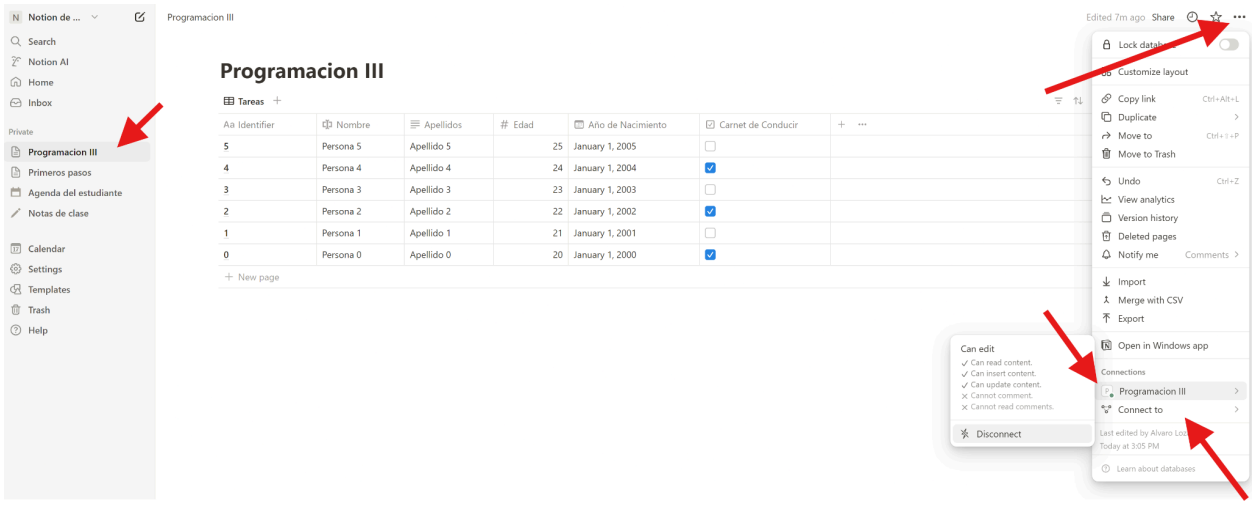


En la imagen anterior es posible ver cómo añadir una nueva property (columna) y determina su tipo (Type). Se cambia a Nombre y de Type Text en este caso.

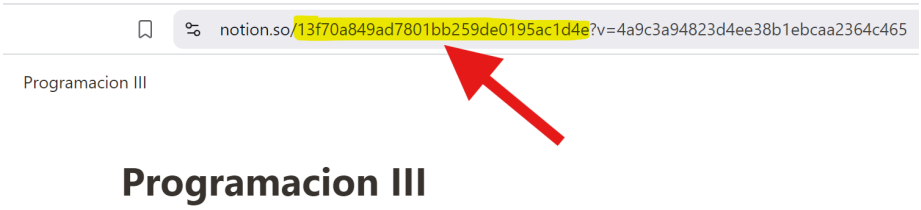
A continuación se muestra un ejemplo con el modelo Persona tratado en el ejemplo proporcionado:



5. A continuación deberemos añadir la integración que se ha creado en el paso 2 a la Page mediante el menú de 3 puntos superior derecho (...). En el apartado connections pulsamos en “Connect to” y seleccionamos la integración que se ha creado en el paso 2 llamada Programación III. Esto dará acceso a dicha integración a esta Page y realizar operaciones CRUD sobre ella.



6. Por último será necesario contar los siguientes parámetros:
   a. **API_KEY** obtenida en el paso 2.
   b. **DATABASE_ID**: esta es posible obtenerla desde la URL de la Page que habéis creado. Es el valor alfanumérico de esta forma: https://www.notion.so/<**DATABASE_ID**>?v=<view_id>



Programacion III

RECOMENDACIONES:

- Procure implementar el resto de la aplicación antes de abordar esta parte.
- Revise el proyecto de ejemplo y consiga hacerlo funcionar en su equipo. Haga modificaciones y familiaricese con el funcionamiento. Si tiene alguna duda la puede plantear en el foro.
- En este modo es necesaria conexión a internet para la interacción con esta API por lo que podrían producirse Excepciones derivadas de falta de conexión o *timeouts*.
- Si no define bien las propiedades de la Database (columnas de la tabla) es probable que reciba errores. Puede cambiar el nivel de mensajes de depuración con la siguiente línea.
- Si tiene problemas en la implementación de esta parte consulte las dudas en el foro de dudas.
- Procure no compartir las API_KEYS en repositorios Github o código. Para ello puede utilizar variables de entorno o proporcionarla como parámetro como se solicita en esta práctica.

REQUISITOS TÉCNICOS Y RECOMENDACIONES

- La aplicación deberá poder ejecutarse en **Linux, Windows y macOS**.
- La aplicación es **multiplataforma** por tanto, no se debe hacer referencia a ningún aspecto de un sistema operativo concreto.
- La aplicación debe presentar una **E/S por consola robusta**.
- Se deben **controlar las posibles excepciones** y gestionarlas correctamente.
- Se deben seguir los **principios de diseño del MVC**, respetando las responsabilidades de cada una de las partes de la aplicación. Los errores restarán puntos en función de su gravedad.
- Se debe utilizar una **sintaxis camelCase**, propia de Java.
- **Revisad los proyectos proporcionados** a lo largo del curso y las indicaciones proporcionadas en este enunciado.
- Si se presentan **dudas**, **dificultades** o necesitáis **aclaraciones** sobre el enunciado, lo mejor es **consultarlas con los profesores de la asignatura**.
- Si se trata de *bugs*, **tras haber invertido un mínimo de tiempo y esfuerzo** en resolverlos, los podéis consultar también al profesorado. **Saber cómo depurar el código es una competencia que hay que adquirir en esta asignatura**.

REQUISITOS DE LA ENTREGA EN STUDIUM **(MUY IMPORTANTE)**

Estos requisitos son obligatorios, la ausencia de alguno de ellos supondrá un 0 suspenso en este trabajo.

- **Fecha límite de entrega: 12 diciembre de 2024 23:59.**
- **Proyecto completo en VSCode**, JDK mínima 21 con las bibliotecas jar incorporadas. Se deberá entregar en un zip con la nomenclatura **Apellido1Apellido2Nombre.zip.**
- **Informe del trabajo en PDF con resultados de la ejecución del programa y unas mínimas explicaciones de cada sección que demuestren la realización del trabajo**. El fichero puede tener el nombre que queráis. Se debe enfocar este informe como un manual técnico que acompaña el software con explicaciones de la funcionalidad y capturas de pantalla del resultado de la ejecución de las funcionalidades del proyecto. Algo que le proporcionaría a otro desarollador. Utilice una plantilla de Google Docs o Word y procure que tenga un buen formato. **Repetimos, el informe es un requisito, sin él no se evalúa la práctica.** No se debe incluir un "*copy paste*" del código en el informe (ya lo estáis subiendo con el proyecto). No debe haber capturas de pantalla sin explicación y no se debe incluir de nuevo el enunciado completo. **No se evalúa al peso, esto busca demostrar que habéis probado el software y que domináis lo que habéis entregado**. Posible esquema de contenidos del informe:

    1. Portada. Nombre,DNI etc.
    2. Introducción: breve explicación
    3. Funcionalidad 1: demostración/captura y explicación
    4. Funcionalidad 2: demostración/captura y explicación
    5. …
    6. Problemáticas, aspectos a mejorar, posibles líneas futuras.

RÚBRICA PARA LA PRÁCTICA

Característica de la aplicación	%	Excelente (9-10)	Notable(7-8)	Suficiente (5-6)	Deficiente (0-4)
CRUD TAREAS **Listados ordenados** **Modificación** **Borrado**	**30%**	El proyecto cumple con los conceptos de diseño MVC y no tiene errores. Pasa todos los test realizados.	El proyecto cumple con los conceptos de diseño del MVC, con algún error puntual.	El proyecto cumple con los conceptos de diseño del MVC, pero presenta algunos errores leves.	El proyecto no cumple con los conceptos de diseño del MVC, o presenta errores graves en la utilización de este patrón.
Exportación Importación CSV	**10%**	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
Exportación Importación JSON	**10%**	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
Serialización	**5%**	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
Parámetros entrada y configuración MVC inicial	**5%**	Se presenta esta característica completamente funcional y sin fallos. Con parámetros y configuración del MVC correcta.	Se utilizan parámetros pero existe algún error puntual.	No se utilizan parámetros pero si se construye el MVC en el main.	No funciona esta característica o presenta errores graves de diseño.
“One more thing” NotionSDK (Uso de bibliotecas externas e integración de código de terceros)	**30%**	Se presenta esta característica completamente funcional y sin fallos.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
Calidad del informe[1]	**10%**	Se presenta un informe detallado, siguiendo las indicaciones de la práctica, mostrando la funcionalidad con capturas y con unas mínimas explicaciones. Además se cuida la presentación, la ortografía, etc.			El informe presentado no tiene capturas o no explica las diferentes funcionalidades de la aplicación. Presenta faltas de ortografía, deficiencias de formato, etc.

[1] **El informe es un requisito**. Si no se entrega no se evaluará la práctica y se considerará como no presentado. Además, el informe tiene su peso en la nota de la práctica ya que demuestra los tests realizados y cómo se ha abordado la práctica.