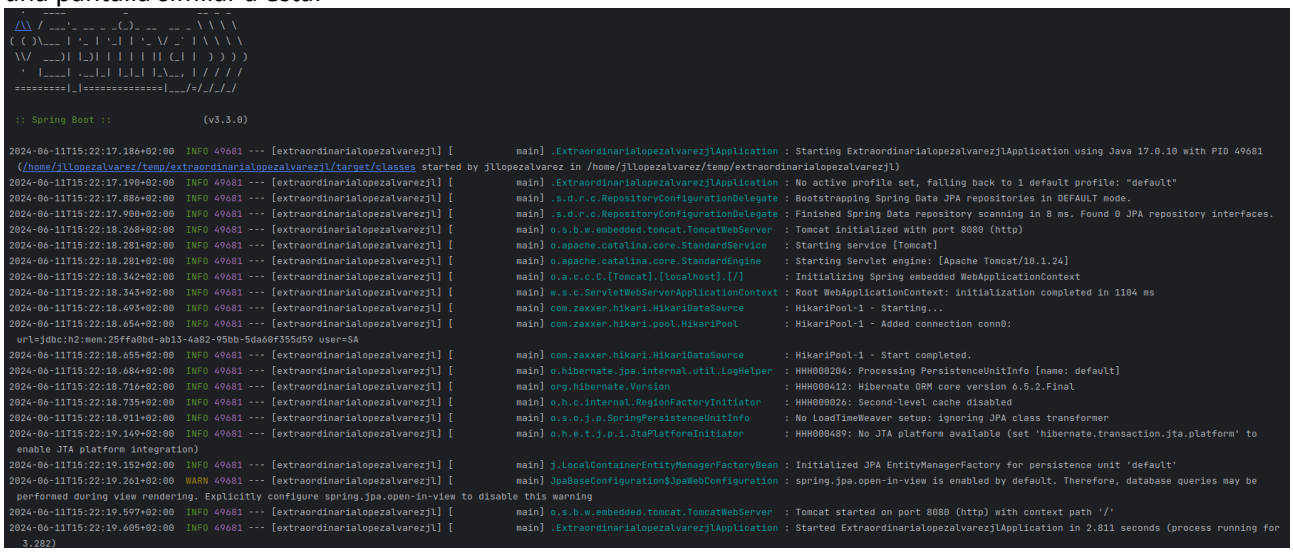
Acceso a datos **Curso 2024/2025**
Simulacro examen 2ª evaluación **Curso: SEGUNDO**
Grupo: DM2E
Fecha: 28/01/2025 **Hora: 15:10 – 17:00**

Creación de proyecto para el examen:
Crear un nuevo proyecto con IntelliJ Idea, con las siguientes características:

- Creado con el asistente de creación de proyectos Spring Initializr
- Nombre del proyecto: simulacro2ev
- Ubicación: a elección del alumno
- Lenguaje y tipo: Java con Maven
- Grupo: es.apellidosnombre (ej:es.lopezalvarezjoseluis)
- Artifact: simulacro2ev
- El nombre del paquete se configurará automáticamente como es.apellidosnombre.simulacro2ev (ej. es.lopezalvarezjl.simulacro2ev)
- Java 17 (preferentemente Amazon Corretto, pero puede usarse otro), empaquetado en Jar
- Versión de Spring Boot: la más moderna que no sea SNAPSHOT o M1
- Dependencias: Spring Web, H2 Database, Spring Data JPA, Lombok (opcional, ya hemos visto en clase que está fallando en ocasiones)

Ejecutar una vez el proyecto para asegurarse de que está correctamente configurado. Puede que sea necesario aceptar la carga del proyecto Maven o habilitar el procesamiento de anotaciones. Debería verse una pantalla similar a esta:

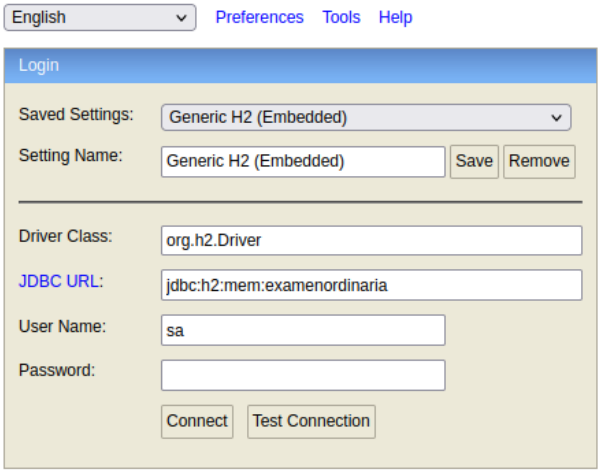


Configurar en el fichero application.properties las siguientes propiedades:

```
spring.datasource.url=jdbc:h2:mem:simulacro2ev;database_to_lower=true;
spring.jpa.defer-datasource-initialization=true
spring.h2.console.enabled=true
```

Volver a arrancar el proyecto y verificar que sigue funcionando tras los cambios en el fichero de configuración.

Al acceder a http://localhost:8080/h2-console debería aparecer algo similar a esto:

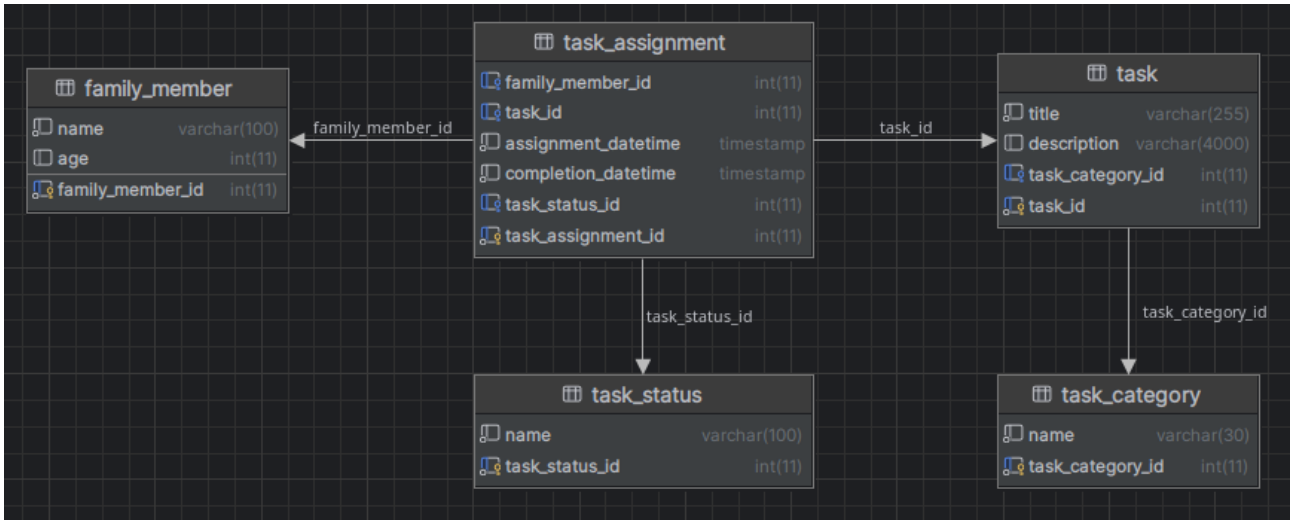


Entrega:

- Se entregará el proyecto completo (incluidos ficheros de configuración como pom.xml, etc.), comprimido en formato zip. El aula virtual no admitirá otros formatos (rar, gzip, 7z, etc.)

Parte 1 – Entidades y modelo de datos – Carga de datos iniciales (3,5 puntos)

Dado el siguiente modelo de base de datos, que representa una serie de tareas domésticas, miembros de una familia, y la asignación de tareas a los miembros de la familia:



Crear las clases Java necesarias para que al ejecutar la aplicación se creen automáticamente las cinco tablas en la base de datos H2. Tener en cuenta las siguientes observaciones / limitaciones:

- Usar todas las anotaciones JPA que se consideren necesarias para cumplir con los requisitos.

- Sobrescribir los métodos equals y hashcode de todas las entidades para que dos objetos se consideren iguales si son de la misma clase si tienen el mismo identificador. Se pueden usar anotaciones Lombok si se ha decidido utilizar esta biblioteca de clases.
- Crear getters / setters según se consideren necesarios. Se pueden usar anotaciones Lombok si se ha decidido usar esta biblioteca de clases.
- Los nombres de tabla / columna generados deben ser los mismos que en el modelo, incluyendo mayúsculas/minúsculas, guiones bajos, etc.:
    - Para conseguir esto no es necesario añadir nada al fichero application.properties, con lo indicado previamente se puede hacer, usando los nombres de atributos / anotaciones adecuadas.
    - Si no se respetan los nombres de tablas y columnas, no se podrá ejecutar el script de carga de datos que se proporciona para facilitar el ejercicio.
- Todas las columnas son obligatorias (not null), salvo:
    - La columna “description” de la tabla “task”
    - La columna “completion_datetime” de la tabla “task_assignment”
- Todas las claves primarias son autonuméricas / autoincrementales / identity.
- Se deben crear las columnas de texto con los tamaños adecuados (30, 100, 255 o 4000).
- En las columnas de tipo “int(11)” se puede ignorar el 11. Usar un tipo de datos “int” o “Integer” en las entidades, sin preocuparse del tamaño.
- Para la columna de tipo “timestamp” usar un atributo de tipo LocalDateTime.
- En cuanto a las relaciones:
    - Todas las tareas tienen, siempre, asociada una categoría de tareas.
    - Todas las asignaciones de tareas se asignan siempre a un miembro de la familia, y siempre tienen un estado.
    - La misma tarea puede estar asignada a varios miembros de la familia. Por ejemplo, “Vacuum the house” (pasar el aspirador) puede estar asignada a la vez a dos miembros de la familia, como “Alice” y “Bob”

Descargar del aula virtual el fichero “data.sql” y colocarlo en la carpeta “resources” del proyecto. Arrancar de nuevo la aplicación y verificar que la inserción de datos iniciales se realiza sin problemas. Verificar que hay datos en la base de datos usando la consola H2. Realizar correcciones **al modelo (no al script)** si fuera necesario para que el script se ejecute correctamente.

Parte 2 – Consulta de datos (3 puntos)

Crear un endpoint de servicio REST para buscar tareas por nombre o descripción, y que devuelva los resultados paginados:
Requisitos:

- Debe usarse una arquitectura en tres capas (controlador / servicio / repositorio)
- El método HTTP debe ser GET
- La URL del endpoint será /api/task/find /{search}/{pageNumber}/{tamanioPagina}, donde:
    - “search” es la cadena de texto que se busca.
    - “pagina” es el número de página. La primera página será la 1.
    - “tamanioPagina” es el tamaño de la página, cuantas tareas se muestran por página.
- Por ejemplo: la URL “/api/task/find/ac/1/3:
    - Busca todas las tareas que contienen “ac” en su nombre o en su descripción (en cualquiera de los dos campos).
    - Pagina los resultados en páginas de 3 elementos, y devuelve la primera página.

- El endpoint devolverá un JSON con la página de tareas solicitada.
- Para cada tarea devolverá:
  - Todos los datos de la tarea: id, título, descripción
  - La categoría de la tarea, un objeto distinto, formado por el id de categoría y su descripción
- Las tareas se devolverán ordenadas por su título.
- Hay libertad para usar métodos disponibles en el repositorio, crear método de consulta derivada, o crear método de consulta personalizada con JPQL o SQL nativo.

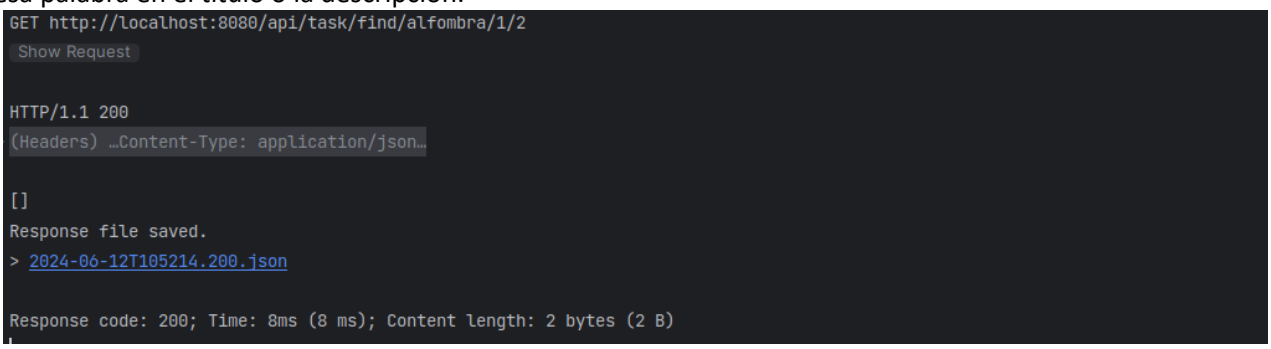
Ejemplo de JSON que se devolvería con los datos cargados con el fichero data.sql:

- Si se lanza la petición /api/task/find/ac/1/2 devuelve el ejemplo de la izquierda, con las dos primeras tareas (la primera página de dos tareas).
- Si se lanza la petición /api/task/find/ac/2/2 devuelve el ejemplo de la derecha, con solo una tarea, porque en total solo hay tres que cumplan el criterio “ac”


```
[
  {
    "taskId": 1,
    "title": "Clean the kitchen",
    "description": "Clean all surfaces and mop the floor",
    "category": {
      "taskCategoryId": 1,
      "name": "Household"
    }
  },
  {
    "taskId": 2,
    "title": "Mow the lawn",
    "description": "Mow the front and backyard lawns",
    "category": {"name": "Outdoor"...}
  },
  {
    "taskId": 7,
    "title": "Vacuum the house",
    "description": "Vacuum all rooms and hallways",
    "category": {"name": "Household"...}
  }
]
```

```
[
  {
    "taskId": 7,
    "title": "Vacuum the house",
    "description": "Vacuum all rooms and hallways",
    "category": {
      "taskCategoryId": 1,
      "name": "Household"
    }
  }
]
```


Si se hace una petición para buscar el texto “alfombra” devolverá un array vacío, porque no hay tareas con esa palabra en el título o la descripción:



Parte 3 – Inserción de datos – 2 puntos

Observar el DTO que se muestra a continuación.

```
@Getter // Si no se usa Lombok, crear los getter de los atributos
@Setter // Si no se usa Lombok, crear los setter de los atributos
public class NewTaskAssignmentDto {
    private int taskId;
    private int familyMemberId;
}
```

Crear un endpoint de servicio REST para, usando este DTO, crear una nueva asignación de tarea.
Requisitos:

- Debe usarse una arquitectura en tres capas (controlador / servicio / repositorio). Crear los controladores / servicios / repositorios que se consideren adecuados.
- El método HTTP debe ser POST
- La URL del endpoint para este servicio será /api/task-assignment.
- El endpoint recibirá en el cuerpo (body) de la petición un objeto NewTaskAssignmentDto. Copiar el código anterior para crear la clase en el proyecto. Crear los getter y setter si no se usa lombok. Si es necesario o se considera adecuado pueden crearse constructores.
- El endpoint devolverá códigos de estado HTTP junto con un mensaje (String):
    - Si se puede crear la asignación de la tarea, devolverá un código 201 (CREATED) con el texto "Asignación creada". Para devolver un 201 con un mensaje se puede usar *return ResponseEntity.status(HttpStatus.CREATED).body(<mensaje>);* Hay otras formas de devolver un 201, es válida cualquiera que sepas hacer.
    - Si no existe la tarea con id "taskId", devolverá un código 404 (NOT FOUND) con el texto "No existe la tarea con id <id>". Para devolver un código 404 con un mensaje se puede usar *return ResponseEntity.status(HttpStatus.NOT_FOUND).body(<mensaje>);* Igual que en el caso anterior, si lo sabes hacer de otra forma, se admite cualquier forma siempre que devuelva un 404.
    - Si no existe el miembro de la familia con id "familyMemberId", devolverá un código 404 (NOT FOUND) con el texto "No existe el miembro familiar con id <id>".
- No es necesario comprobar que la tarea se ha asignado previamente al miembro de la familia. Admitimos que la misma tarea se asigne múltiples veces al mismo miembro familiar.
- Al crear la asignación de la tarea, se asigna el siguiente valor a los atributos que no se reciben en el DTO:
    - taskAssignmentId: como este campo es la clave primaria y es autoincremental, debe asignarlo automáticamente la base de datos.
    - assignmentDatetime: la fecha y hora actual.
    - completionDatetime: null, porque no se ha terminado la tarea.
    - status: debe asociarse a un objeto TaskStatus con valor "pending" (taskStatusId = 1).
- No se pueden escribir métodos de repositorio específicos para esta tarea. Se deben usar sólo métodos ofrecidos por la interfaz heredada en el repositorio (típicamente CrudRepository o JpaRepository).

Por ejemplo, si se hiciera la siguiente petición nada más arrancar la aplicación:

```
POST http://localhost:8080/api/task-assignment
Content-Type: application/json
{
  "familyMemberId": 1,
  "taskId": 1
}
```

Devolvería un código 201:

```
POST http://localhost:8080/api/task-assignment
HTTP/1.1 201
Content-Type: text/plain;charset=UTF-8
Content-Length: 18
Date: Wed, 12 Jun 2024 11:21:49 GMT
Keep-Alive: timeout=60
Connection: keep-alive

Asignación creada
```

Y en la BD debe haber un nuevo registro para la asignación de la tarea.

Si la petición usa un código de tarea inexistente (100, por ejemplo) devolverá un 404:

```
POST http://localhost:8080/api/task-assignment
HTTP/1.1 404
Content-Type: text/plain;charset=UTF-8
Content-Length: 29
Date: Wed, 12 Jun 2024 11:32:58 GMT
Keep-Alive: timeout=60
Connection: keep-alive

No existe la tarea con id 100
```

Si la petición usa un código de miembro familiar inexistente (100, por ejemplo) también devolverá un 404, pero con el mensaje “No existe el miembro familiar con id 100”

Parte 4 – Actualización de datos – 1,5 puntos

Crear un endpoint de servicio REST para indicar la finalización de una tarea.
Requisitos:

- Debe usarse una arquitectura en tres capas (controlador / servicio / repositorio)
- El método HTTP debe ser PUT
- La URL del endpoint para este servicio será /api/task-assignmet/complete/<id de la asignación>. Por ejemplo, para finalizar la asignación de tarea con el código 50, se realizará una petición PUT a /api/task-assignment/complete/50.
- El endpoint devolverá un código de respuesta con un mensaje indicando el éxito o no de la operación. En concreto:
    - Devolverá un 404 (NOT_FOUND) si no se encuentra la asignación de tarea en la BBDD, con el mensaje "No se encuentra la asignación de tarea con id <id>"
    - Devolverá un 400 (BAD_REQUEST) si la tarea ya está completada, con el mensaje "La tarea con id <id> ya ha sido completada"
    - Devolverá un 200 (OK) si se ha podido marcar la tarea como completada, con el mensaje "Tarea completada". En este caso, actualizará el registro de la asignación en la BD, estableciendo el estado en completado (valor 3) y la fecha del campo "completed_datetime" a la fecha y hora actual.
- No se pueden escribir métodos de repositorio específicos para esta tarea. Se deben usar sólo métodos ofrecidos por la interfaz heredada en el repositorio (típicamente CrudRepository o JpaRepository).

Por ejemplo, si se hiciera la siguiente petición nada más arrancar la aplicación, para completar la asignación de tarea con id 1000, que no existe, devolvería un 404:

```
POST http://localhost:8080/api/task-assignment/complete/1000
HTTP/1.1 404
Content-Type: text/plain;charset=UTF-8
Content-Length: 45
Date: Wed, 12 Jun 2024 12:08:38 GMT
Keep-Alive: timeout=60
Connection: keep-alive

No existe la asignación de tarea con id 1000
```

Si se hace la misma petición con el id 1, que sí existe y no está completada, devuelve un 200:

```
POST http://localhost:8080/api/task-assignment/complete/1
HTTP/1.1 200
Content-Type: text/plain;charset=UTF-8
Content-Length: 16
Date: Wed, 12 Jun 2024 12:10:38 GMT
Keep-Alive: timeout=60
Connection: keep-alive

Tarea completada
```

Y si repetimos la petición, como ya está completada la tarea, devolverá un 400.

```
POST http://localhost:8080/api/task-assignment/complete/1
HTTP/1.1 400
Content-Type: text/plain;charset=UTF-8
Content-Length: 54
Date: Wed, 12 Jun 2024 12:14:37 GMT
Connection: close

La asignación de tarea con id 1 ya ha sido completada
```