**Universidad Autónoma de Yucatán**

**Licenciatura en Ingeniería de Software**

**Documentación de los componentes**

**Preparado por:**

Irving Báez Córdova,

Daniel Medina Cámara,

Moisés Vinajera Sosa,

Gerardo Caamal

**28 de noviembre de 2021**

**Índice**

[**1. Descripción de Funcionalidades** 4](#_Toc89678274)

[**1.1 Modulo – Student** 4](#_Toc89678275)

[Objetivo Principal: Desarrollar una aplicación CRUD para la gestión de los estudiantes junto con la definición de las interfaces de comunicación entre el cliente y el sistema backend así mismo con el manejo de la información en una base de datos. 4](#_Toc89678276)

[**BackEnd** 4](#_Toc89678277)

[Funcionalidades: 4](#_Toc89678278)

[**FrontEnd** 4](#_Toc89678279)

[Funcionalidades: 4](#_Toc89678280)

[**2. Descripción de Componentes** 5](#_Toc89678281)

[**BackEnd** 5](#_Toc89678282)

[**2.1 Componente - Student** 5](#_Toc89678283)

[**2.2 Componente – StudentController** 5](#_Toc89678284)

[**2.3 Componente – StudentService** 7](#_Toc89678285)

[**2.4 Componente – StudentRepository** 8](#_Toc89678286)

[**FrontEnd** 9](#_Toc89678287)

[**2.5 Componente – Client** 9](#_Toc89678288)

[**2.6 Componente – App** 9](#_Toc89678289)

[**2.7 Componente – StudentDrawerForm** 10](#_Toc89678290)

[**2.8 Componente – StudentEditDrawerForm** 11](#_Toc89678291)

[**2.9 Diagrama de componentes** 12](#_Toc89678292)

[**2. Descripción de Clases** 12](#_Toc89678293)

[**BackEnd** 12](#_Toc89678294)

[**3.1 Clase - Student** 12](#_Toc89678295)

[**3.2 Clase - StudentController** 13](#_Toc89678296)

[**3.3 Clase - StudentService** 15](#_Toc89678297)

[**3.4 Interfaz - StudentRepository** 16](#_Toc89678298)

[**4. Descripción de las secuencias** 18](#_Toc89678299)

[**4.1 Diagrama de secuencia - getAllStudents** 18](#_Toc89678300)

[**4.2 Diagrama de secuencia - addNewStudent** 18](#_Toc89678301)

[**4.3 Diagrama de secuencia - addNewStudentThrowBadRequestException** 19](#_Toc89678302)

[**4.4 Diagrama de secuencia - updateStudent** 19](#_Toc89678303)

[**4.5 Diagrama de secuencia - deleteStudent** 20](#_Toc89678304)

[**4.6 Diagrama de secuencia - deleteStudentsByIds** 20](#_Toc89678305)

[**5. Descripción de caso de uso** 21](#_Toc89678306)

# **1. Descripción de Funcionalidades**

# **1.1 Modulo – Student**

# Objetivo Principal: Desarrollar una aplicación CRUD para la gestión de los estudiantes junto con la definición de las interfaces de comunicación entre el cliente y el sistema backend así mismo con el manejo de la información en una base de datos.

# **BackEnd**

## Funcionalidades:

* Representar la entidad estudiante en una clase.
* Mapear la entidad Student a una tabla en la base de datos
* Contiene las validaciones para los atributos de la clase Student
* Asigna los servicios requeridos por las peticiones del cliente (obtener todos los estudiantes, agregar un nuevo estudiante, editar un nuevo estudiante y eliminar un estudiante).
* Envía la respuesta de los servicios al cliente.
* Se comunica con la capa de acceso a la base de datos.
* Contiene la lógica de negocios para obtener todos los estudiantes, agregar un nuevo estudiante, editar un nuevo estudiante y eliminar un estudiante.
* Maneja las excepciones
* Recupera información de la base de datos
* Recupera a los estudiantes de la base de datos
* Guarda estudiantes nuevos de la base de datos
* Actualiza la información de los estudiantes de la base de datos
* Elimina los estudiantes de la base de datos

# **FrontEnd**

## Funcionalidades:

* Envía las peticiones de obtener todos los estudiantes al sistema BackEnd, agregar un nuevo estudiante, editar un estudiante y eliminar un nuevo estudiante.
* Captura y depura las excepciones recibidas por el sistema Backend.
* Despliega las componentes de la interfaz gráfica.
* Despliega los estudiantes en una tabla general.
* Despliega el componente gráfico para agregar un nuevo estudiante.
* Despliega el componente gráfico para editar un estudiante existente.

## **2. Descripción de Componentes**

# **BackEnd**

# **2.1 Componente - Student**

**Nombre:** Student

**Descripción:** Representa a la entidad de los estudiantes, se mapea a una tabla en la base de datos y contiene las validaciones de los atributos de la clase Student.

Dependencias con otros componentes: Ninguna

**Interfaces de Salida:**

* getId -Devuelve el atributo id de la clase Student
* setId – Modifica el atributo id de la clase Student
* getName -Devuelve el atributo name de la clase Student
* setName – Modifica el atributo name de la clase Student
* getEmail -Devuelve el atributo email de la clase Student
* setEmail – Modifica el atributo email de la clase Student
* getGender -Devuelve el atributo gender de la clase Student
* setGender – Modifica el atributo gender de la clase Student
* getAge -Devuelve el atributo age de la clase Student
* setAge – Modifica el atributo age de la clase Student

**Interfaces de Entrada:** Ninguna

**Artefactos:**

* Biblioteca javax.persistence;
* Biblioteca javax.validation.

## **2.2** **Componente – StudentController**

**Nombre:** StudentController

**Descripción:** Provee acceso al comportamiento del sistema backend por medio de la asignación de servicios específicos a partir de las peticiones que llegan a través del cliente y verifica que los datos enviados por el cliente sean válidos.

**Dependencias con otros componentes:**

* Student
* StudentService

**Interfaces de Salida:**

* getAllStudents – Intercepta las peticiones GET en el endpoint "api/v1/students" y asigna al servicio específico (StudentService - getAllStudents) para devolver todos los estudiantes almacenados en la base de datos
* addStudent - Intercepta las peticiones POST en el endpoint "api/v1/students" y asigna al servicio específico (StudentService - addStudent) para guardar un nuevo estudiante en la base de datos.
* updateStudent - Intercepta las peticiones PUT en el endpoint "api/v1/students/{studentId}" y asigna al servicio específico (StudentService - updateStudent) para actualizar los datos de un estudiante existente en la base de datos.
* deleteStudent - Intercepta las peticiones DELETE en el endpoint "api/v1/students/{studentId}" y asigna al servicio específico (StudentService - deleteStudent) para eliminar un estudiante de la base de datos.
* deleteSelectStudents - Intercepta las peticiones DELETE en el endpoint "api/v1/students/deletestudents" y asigna al servicio específico (StudentService - deleteSelectStudents) para eliminar una serie de estudiante seleccionados de la base de datos.

**Interfaces de Entrada:**

* StudentService – getAllStudents – Provee una lista con todos los estudiantes almacenados en la base de datos.
* StudentService – addStudent – Guarda un nuevo estudiante a la base de datos.
* StudentService – updateStudent – Actualiza un estudiante existente a la base de datos.
* StudentService – deleteStudent – Elimina un estudiante de la base de datos
* StudentService – deleteSelectStudents – Elimina a los estudiantes seleccionados de la base de datos.

**Artefactos**

* Biblioteca javax.validation
* Interfaz @RestController
* Interfaz @RequestMapping

## **2.3 Componente – StudentService**

**Nombre:** StudentService

**Descripción:** Contiene la lógica del negocio. Se comunica con la capa de acceso a la base de datos. Contiene la lógica de para obtener todos los estudiantes, agregar un nuevo estudiante, editar un nuevo estudiante y eliminar un estudiante. Lanza una excepción (BadRequestException) si el email de un nuevo estudiante ya está registrado. Lanza una excepción (StudentNotFoundException) si el estudiante no se encontró en la base de datos.

**Dependencias con otros componentes:**

* Student
* StudentRepository

**Interfaces de Salida:**

* getAllStudents – Provee una lista con todos los estudiantes almacenados en la base de datos.
* addStudent – Guarda un nuevo estudiante a la base de datos.
* updateStudent – Actualiza un estudiante existente a la base de datos.
* deleteStudent – Elimina un estudiante de la base de datos
* deleteSelectStudents – Elimina a los estudiantes seleccionados de la base de datos.

**Interfaces de Entrada:**

* StudentRepository – findAll – Recupera todos los estudiantes de la base de datos.
* StudentRepository – selectExistEmail – Verifica si un email está actualmente registrado en la base de datos.
* StudentRepository – save – Guarda un nuevo estudiante o actualiza un estudiante existente en la base de datos.
* StudentRepository – existsById – Verifica si existe un estudiante en la base de datos a partir de su id.
* StudentRepository – deleteById – Elimina un estudiante de la base de datos a partir de su id.
* StudentRepository – findById – Devuelve un estudiante de la base de datos a partir de su id.
* StudentRepository – deleteByIdIn – Elimina todos los estudiantes seleccionados de la base de datos a partir de una serie de ids almacenados en una lista.

**Artefactos:**

* Artefacto org.springframework.stereotype.Service
* Artefacto org.springframework.transaction.annotation.Transactional
* Excepción BadRequestException;
* Excepción StudentNotFoundException;

## **2.4 Componente – StudentRepository**

Nombre: StudentRepository

Descripción: Representa a la capa de persistencia con la que se accede a la base de datos. Es la entidad encargada de comunicarse directamente con la base de datos. Es la capa del ORM.

• Recupera a los estudiantes de la base de datos

• Guarda estudiantes nuevos de la base de datos

• Actualiza la información de los estudiantes de la base de datos

• Elimina los estudiantes de la base de datos

• Verifica si el email de un estudiante nuevo ya está registrado en la base de datos

Dependencias con otros componentes:

* Hereda de la interfaz JpaRepository
* Student

Interfaces de Salida:

* findAll – Recupera todos los estudiantes de la base de datos.
* selectExistEmail – Verifica si un email está actualmente registrado en la base de datos.
* save – Guarda un nuevo estudiante o actualiza un estudiante existente en la base de datos.
* existsById – Verifica si existe un estudiante en la base de datos a partir de su id.
* deleteById – Elimina un estudiante de la base de datos a partir de su id.
* findById – Devuelve un estudiante de la base de datos a partir de su id.
* deleteByIdIn – Elimina todos los estudiantes seleccionados de la base de datos a partir de una serie de ids almacenados en una lista.

Interfaces de Entrada:

* jdbc:postgresql://localhost:5433/amigoscode - es el canal de comunicación con la base de datos.

Artefactos:

* Artefacto org.springframework.stereotype.Repository
* Artefacto org.springframework.data.jpa.repository.JpaRepository;

# **FrontEnd**

# **2.5 Componente – Client**

**Nombre:** Client

**Descripción:** Maneja y envía las peticiones del cliente a partir de los endpoints declarados por el sistema backend y recibe las respuestas de este. Enviar las peticiones de obtener todos los estudiantes, agregar un nuevo estudiante, editar un estudiante y eliminar un nuevo estudiante. Captura y depura las excepciones recibidas por el sistema backend.

**Dependencias con otros componentes:**

* Endpoints del sistema BackEnd

**Interfaces de Salida:**

* getAllStudents – hace una petición GET al endpoint api/v1/students para obtener a todos los estudiantes.
* addNewStudent – hace una petición POST al endpoint api/v1/students para guardar un nuevo estudiante.
* deleteStudent - hace una petición DELETE al endpoint api/v1/students/ {studentId} para eliminar un estudiante.
* putStudent - hace una petición PUT al endpoint api/v1/students/ {studentId} para actualizar un estudiante.
* deleteSelectStudents - hace una petición DELETE al endpoint api/v1/students/deletestudents para eliminar los estudiantes seleccionados.
* checkStatus – verifica que las peticiones retornen un status http 200 con la respuesta asociada o las excepciones en caso de ser necesarias.

**Interfaces de Entrada:**

* GET - api/v1/students – Retorna todos los estudiantes
* POST - api/v1/students – Guarda un nuevo estudiante.
* PUT - api/v1/{studentId} – Actualiza un estudiante existente.
* DELETE - api/v1/{studentId} – Elimina un estudiante existente.
* DELETE - api/v1/deletestudents– Elimina los estudiante seleccionados.

**Artefactos:**

* Biblioteca - unfetch

## **2.6 Componente – App**

**Nombre:** App

**Descripción:** Despliega los componentes de la interfaz gráfica así mismo la información recibida desde el sistema BackEnd. Se comunica con el componente Client para solicitar información del sistema BackEnd. Maneja las excepciones capturadas por el cliente y lanza notificaciones de los errores. Despliega los estudiantes en una tabla general.

**Dependencias con otros componentes:**

* Client
* StudentDrawerForm
* StudentEditDrawerForm

**Interfaces de Salida:**

* removeStudent – Se comunica con el componente Client para eliminar un estudiante, manda una notificación de éxito o una notificación de error.
* fetchStudents – Guarda los resultados recibidos por el Client en formato json. Envía una notificación de error en caso de que la petición no sea exitosa.
* updateStudent – Despliega el componente gráfico para ingresar los datos que se quieren actualizar de un estudiante. Guarda el estudiante que se quiere editar.
* renderStudent – Retorna los componentes de la table de estudiante, el componente para dar de alta un nuevo estudiante y el componente para editar un estudiante.
* deleteStudentsById - Se comunica con el componente Client para eliminar los estudiantes seleccionados, manda una notificación de éxito o una notificación de error.

**Interfaces de Entrada:**

* Client getAllStudents – hace una petición GET al endpoint api/v1/students para obtener a todos los estudiantes.
* Client - deleteStudent - hace una petición DELETE al endpoint api/v1/students/ {studentId} para eliminar un estudiante.
* Client – deleteSelectStudents - hace una petición DELETE al endpoint api/v1/students/deletestudents para eliminar los estudiantes seleccionados.

**Artefactos:**

* Biblioteca – React
* Biblioteca -Antd
* Biblioteca - @ant-design/icons

## **2.7 Componente – StudentDrawerForm**

**Nombre:** StudentDrawerForm

**Descripción:** Desplegar un componente gráfico para el ingreso de la información de un nuevo estudiante.

**Dependencias con otros componentes:**

* Client

**Interfaces de Salida:**

* onClose – guarda el componente StudentDrawerForm
* onFinish – Guarda un nuevo estudiante con los datos que se ingresaron. Envía una notificación de existo en caso de que sea exitosa o una notificación de error en caso contrario.

**Interfaces de Entrada:**

* Client - addNewStudent – hace una petición POST al endpoint api/v1/students para guardar un nuevo estudiante.

**Artefactos:**

* Biblioteca – React
* Biblioteca -Antd
* Biblioteca - @ant-design/icons

## **2.8 Componente – StudentEditDrawerForm**

**Nombre:** StudentEditDrawerForm

**Descripción:** Desplegar un componente gráfico para el ingreso actualizar la información de un estudiante.

**Dependencias con otros componentes:**

* Client

**Interfaces de Salida:**

* onClose – guarda el componente StudentEditDrawerForm
* onFinish – Actualiza la información de un estudiante con los datos que se ingresaron. Envía una notificación de existo en caso de que sea exitosa o una notificación de error en caso contrario.

**Interfaces de Entrada:**

* Client - putStudent - hace una petición PUT al endpoint api/v1/students/ {studentId} para actualizar un estudiante.

**Artefactos:**

* Biblioteca – React
* Biblioteca -Antd
* Biblioteca - @ant-design/icons

## **2.9 Diagrama de componentes**

## **2. Descripción de Clases**

# **BackEnd**

# **3.1 Clase - Student**

**Nombre de la clase:** Student

**Descripción:** Representa a la entidad de los estudiantes, se mapea a una tabla en la base de datos y contiene las validaciones de los atributos de la clase.

**Dependencias con otras clases:**

* StudentController – El controlador recibe un objeto Student como parámetro de entrada para las funciones addStudent y updateStudent.
* StudentService - El servicio recibe un objeto Student como parámetro de entrada para las funciones addStudent y updateStudent.
* StudentRepository – El repositorio utiliza la entidad Student para mapear los objetos a la base de datos.

**Atributos:**

* id
  + Tipo: Long
  + Visibilidad: private
  + Descripción: Representa al identificador del estudiante
* name
  + Tipo: String
  + Visibilidad: private
  + Descripción: Representa el nombre del estudiante
* gender
  + Tipo: Gender
  + Visibilidad: private
  + Descripción: Es un objeto enum tipo String. Representa el género del estudiante.
* email
  + Tipo: String
  + Visibilidad: private
  + Descripción: Representa el email del estudiante.
* age
  + Tipo: Integer
  + Visibilidad: private
  + Descripción: Representa la edad del estudiante

**Funciones:**

* Getters
* Setters
* ToString
* Equals
* HashCode

# **3.2 Clase - StudentController**

**Nombre de la clase:** StudentController

**Descripción:** Provee las interfaces de acceso al comportamiento del sistema, asigna servicios específicos a partir de las peticiones que llegan a través del cliente y verifica que los datos enviados por el cliente sean válidos.

**Dependencias con otras clases:**

* Student – Utiliza la entidad Student como parámetros en las funciones addStudent y updateStudent.
* StudentService – Utiliza los servicios que provee la clase StudentService para retornar los valores esperados al cliente.

**Atributos:**

* studentService
  + Tipo: StudentService
  + Visibilidad: private final
  + Valor por omisión: El framework crea la instancia al ejecutar y la inyecta cuando se requiere.
  + Descripción: Provee la lógica de negocio. Los servicios que provee son obtener todos los estudiantes, agregar un nuevo estudiante, actualizar un estudiante existente, eliminar un estudiante y eliminar estudiantes seleccionados.

**Funciones:**

* getAllStudents
  + Argumentos: void
  + Valor de retorno: List<Student>
  + Visibilidad: Public
  + Servicio: Obtener todos los estudiantes.
  + Interfaz de salida: “api/v1/students”
  + Descripción: Intercepta las peticiones GET en la interfaz de salida y llama al servicio requerido (StudentService - getAllStudents) y devuelve el resultado esperado.
* addStudent
  + Argumentos: Student student
  + Valor de retorno: void
  + Visibilidad: Public
  + Servicio: Agregar un nuevo estudiante.
  + Interfaz de salida: “api/v1/students”
  + Descripción: Intercepta las peticiones POST en la interfaz de salida y llama al servicio requerido (StudentService – addStudent) y devuelve el resultado esperado.
* updateStudent
  + Argumentos: Long StudentId, Student student
  + Valor de retorno: void
  + Visibilidad: Public
  + Servicio: Actualizar un estudiante.
  + Interfaz de salida: “api/v1/students/{studentId}”
  + Descripción: Intercepta las peticiones PUT en la interfaz de salida y llama al servicio requerido (StudentService – updateStudent) y devuelve el resultado esperado.
* deleteStudent
  + Argumentos: Long StudentId
  + Valor de retorno: void
  + Visibilidad: Public
  + Servicio: Elimina un estudiante.
  + Interfaz de salida: “api/v1/students/{studentId}”
  + Descripción: Intercepta las peticiones DELETE en la interfaz de salida y llama al servicio requerido (StudentService – deleteStudent) y devuelve el resultado esperado.
* deleteSelectStudents
  + Argumentos: List<Long> ids
  + Valor de retorno: void
  + Visibilidad: Public
  + Servicio: Elimina los estudiantes seleccionados.
  + Interfaz de salida: “api/v1/students/deletestudents”
  + Descripción: Intercepta las peticiones DELETE en la interfaz de salida y llama al servicio requerido (StudentService – deleteSelectStudents) y devuelve el resultado esperado.

# **3.3 Clase - StudentService**

**Nombre de la clase:** StudentService

**Descripción:** Contiene la lógica del negocio. Se comunica con la capa de acceso a la base de datos. Contiene la lógica de para obtener todos los estudiantes, agregar un nuevo estudiante, editar un nuevo estudiante y eliminar un estudiante. Lanza una excepción (BadRequestException) si el email de un nuevo estudiante ya está registrado.

**Dependencias con otras clases:**

* StudentRepository – Utiliza a la entidad StudentRepository para acceder a la base de datos.

**Atributos:**

* studentRepository
  + Tipo: StudentRepository
  + Visibilidad: private final
  + Valor por omisión: El framework crea la instancia al ejecutar y la inyecta cuando se requiere.
* Descripción: Provee el acceso a la base de datos.

**Funciones:**

* getAllStudents
  + Argumentos: void
  + Valor de retorno: List<Student>
  + Visibilidad: Public
  + Servicio: Obtener todos los estudiantes.
  + Interfaz de salida: getAllStudents()
  + Descripción: Se comunica con la capa de repositorio para obtener a los estudiantes.
* addStudent
  + Argumentos: Student student
  + Valor de retorno: void
  + Visibilidad: Public
  + Servicio: Agregar un nuevo estudiante.
  + Interfaz de salida: addStudent()
  + Descripción: Se comunica con la capa de repositorio para guardar un nuevo estudiante, lanza una excepción(BadRequestException) si el email ya existe en la base de datos.
* updateStudent
  + Argumentos: Long StudentId, Student student
  + Valor de retorno: void
  + Visibilidad: Public
  + Servicio: Actualizar un estudiante.
  + Interfaz de salida: updateStudent()
  + Descripción: Se comunica con la capa de repositorio para actualizar un estudiante, lanza una excepción (StudentNotFoundException) si no se encuentra el estudiante en la base de datos.
* deleteStudent
  + Argumentos: Long StudentId
  + Valor de retorno: void
  + Visibilidad: Public
  + Servicio: Elimina un estudiante.
  + Interfaz de salida: deleteStudent()
  + Descripción: Se comunica con la capa de repositorio para eliminar un estudiante, lanza una excepción (StudentNotFoundException) si no se encuentra el estudiante en la base de datos.
* deleteSelectStudents
  + Argumentos: List<Long> ids
  + Valor de retorno: void
  + Visibilidad: Public
  + Servicio: Elimina los estudiantes seleccionados.
  + Interfaz de salida: deleteSelectStudents()
  + Descripción: Se comunica con la capa de repositorio para eliminar los estudiantes seleccionados.

# **3.4 Interfaz - StudentRepository**

**Nombre de la clase:** StudentRepository

**Descripción:** Representa a la capa de persistencia con la que se accede a la base de datos. Es la entidad encargada de comunicarse directamente con la base de datos. Es la capa del ORM.

**Dependencias con otras clases:**

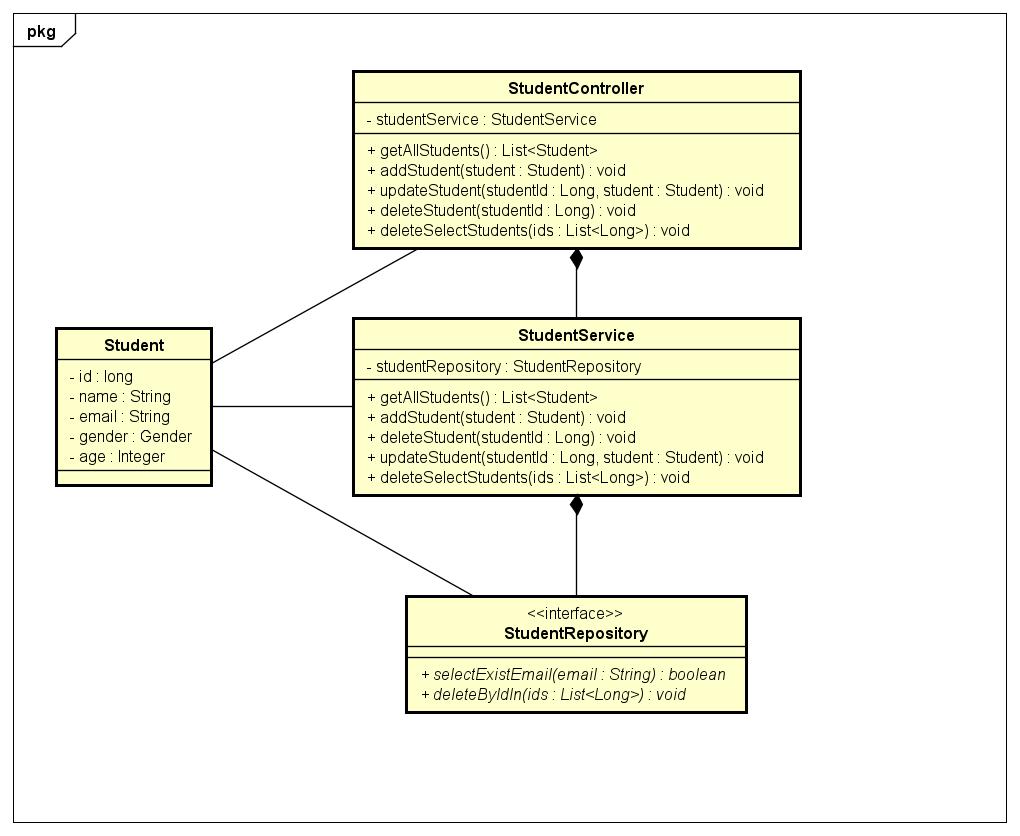
* JpaRepository – Hereda las funciones de la clase mencionada
* Student - utiliza la entidad Student para mapear los objetos a la base de datos.

**Atributos:** Ninguno

**Funciones:**

* selectExistsEmail
  + Argumentos: String email
  + Valor de retorno: boolean
  + Visibilidad: default
  + Servicio: Verifica que existe un email en la base de datos
  + Interfaz de salida: selectExistsEmail()
  + Descripción: Se comunica con la base de datos para verificar si existe un email.
* deleteByIdIn
  + Argumentos: List<Long> ids
  + Valor de retorno: void
  + Visibilidad: default
  + Servicio: Elimina a los estudiantes con los ids seleccionados
  + Interfaz de salida: deleteByIdIn()
  + Descripción: Se comunica con la base de datos para eliminar los estudiantes seleccionados por el id.

**3.5 Diagrama de clases**

****

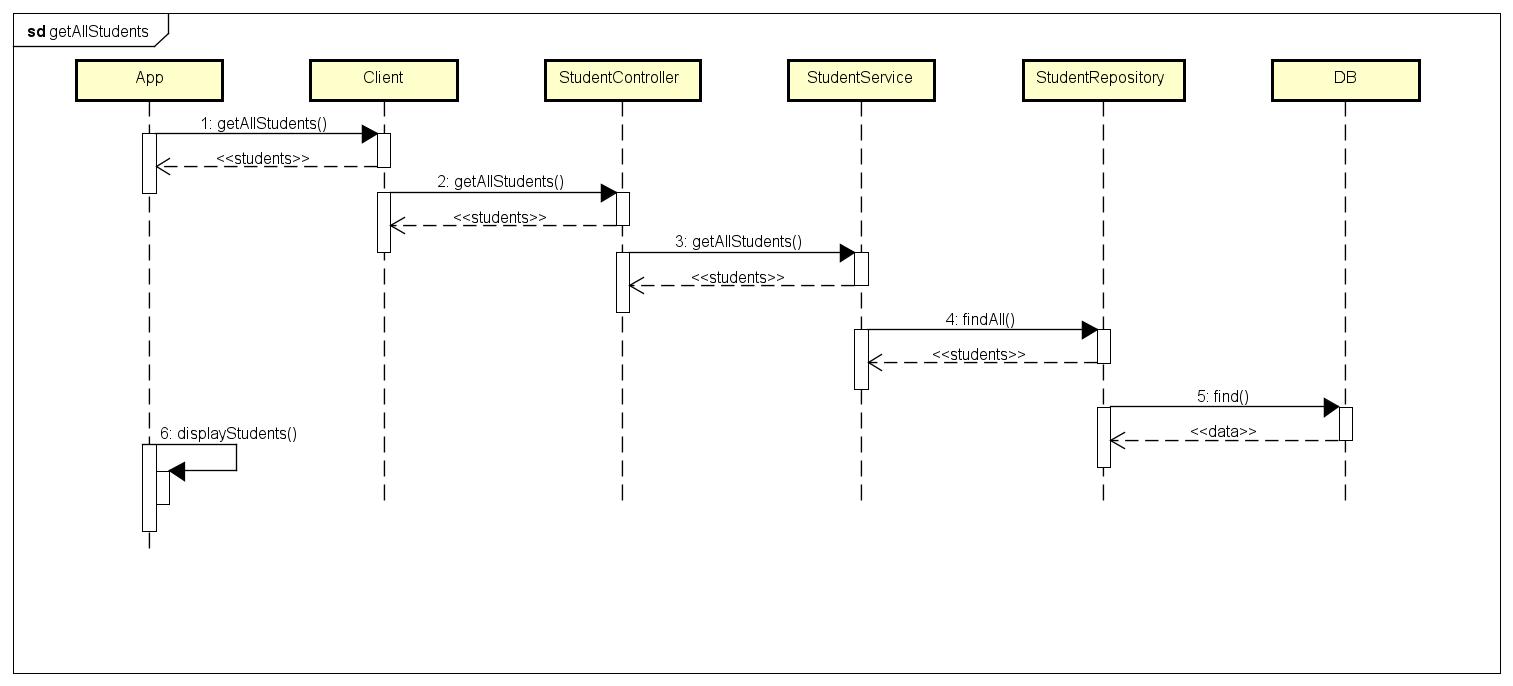
# **4. Descripción de las secuencias**

# **4.1 Diagrama de secuencia -** **getAllStudents**

**Nombre:** getAllStudents

**Descripción:** Muestra la secuencia desde que el cliente(frontend) pide a todos los alumnos al sistema BackEnd

**Diagrama:**

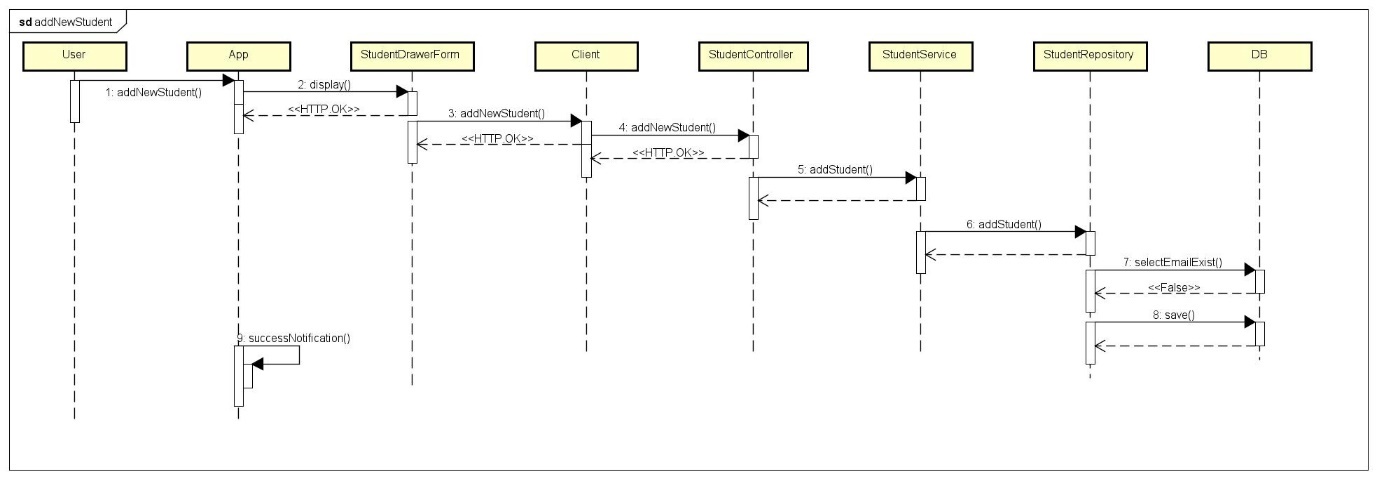


# **4.2 Diagrama de secuencia - addNewStudent**

**Nombre:** addNewStudent

**Descripción:** Muestra la secuencia desde que el cliente(frontend) ingresa los datos de un nuevo estudiante al sistema BackEnd, se verifica que el email no exista, se registra a la base de datos, retorna un status HTTP 200 OK y despliega una notificación de éxito.

**Diagrama:**

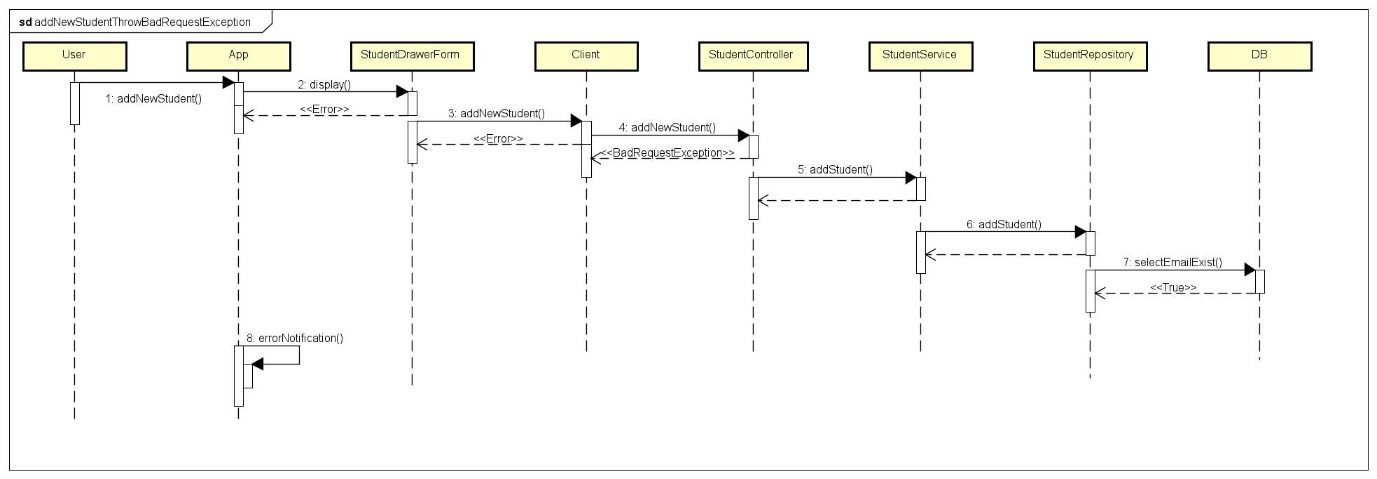


# **4.3 Diagrama de secuencia - addNewStudentThrowBadRequestException**

**Nombre:** addNewStudentThrowBadRequestException

**Descripción:** Muestra el flujo de secuencias desde que el cliente(frontend) ingresa los datos de un nuevo estudiante al sistema BackEnd, se verifica que el email existe y se lanza una excepción BadRequestException.

**Diagrama:**

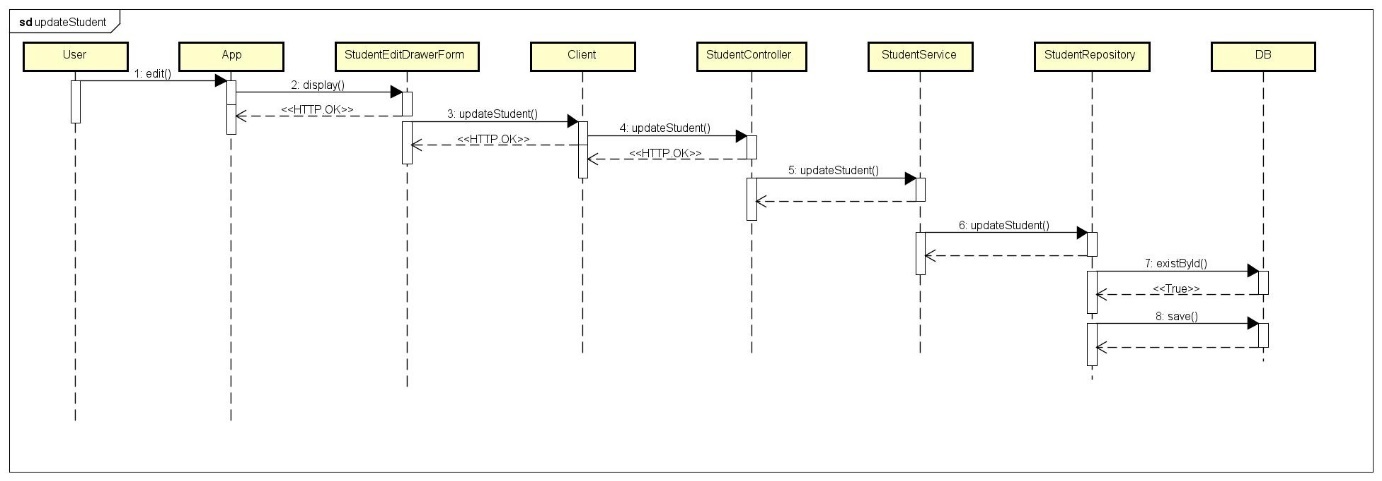
****

# **4.4 Diagrama de secuencia - updateStudent**

**Nombre:** updateStudent

**Descripción:** Muestra la secuencia desde que el cliente(frontend) ingresa los datos para actualizar un estudiante al sistema BackEnd, se verifica que el estudiante exista, se actualiza en la base de datos, retorna un status HTTP 200 OK.

**Diagrama:**

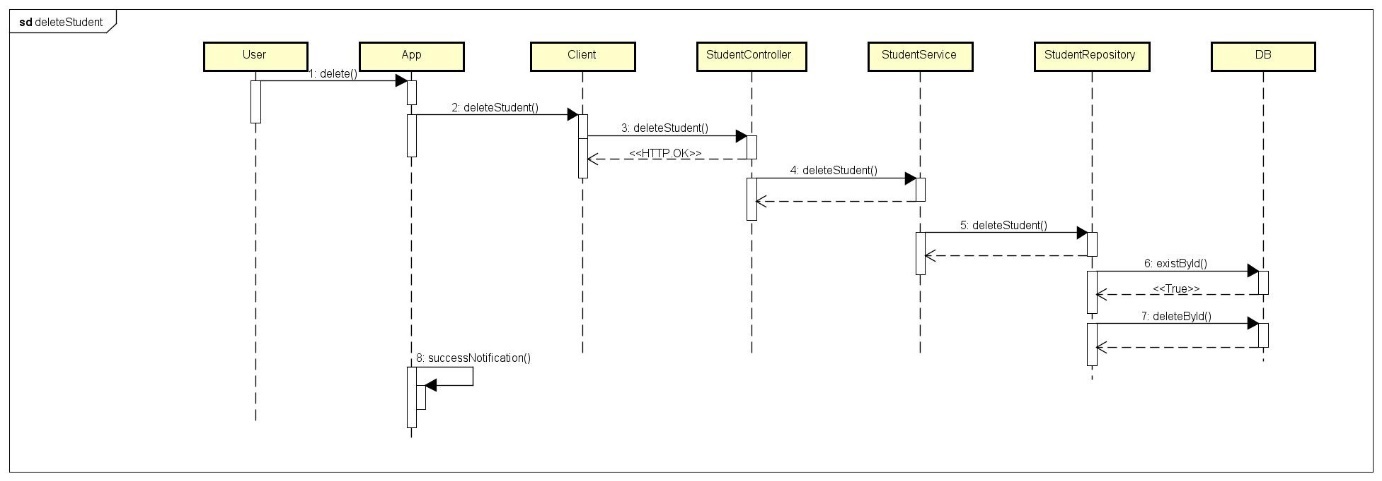
****

# **4.5 Diagrama de secuencia - deleteStudent**

**Nombre:** deleteStudent

**Descripción:** Muestra la secuencia desde que el cliente(frontend) selecciona eliminar estudiante, se hace la petición Delete al sistema BackEnd, se verifica que el estudiante existe, se elimina de la base de datos, se lanza un status HTTP 200 y se despliega una notificación de éxito.

**Diagrama:**

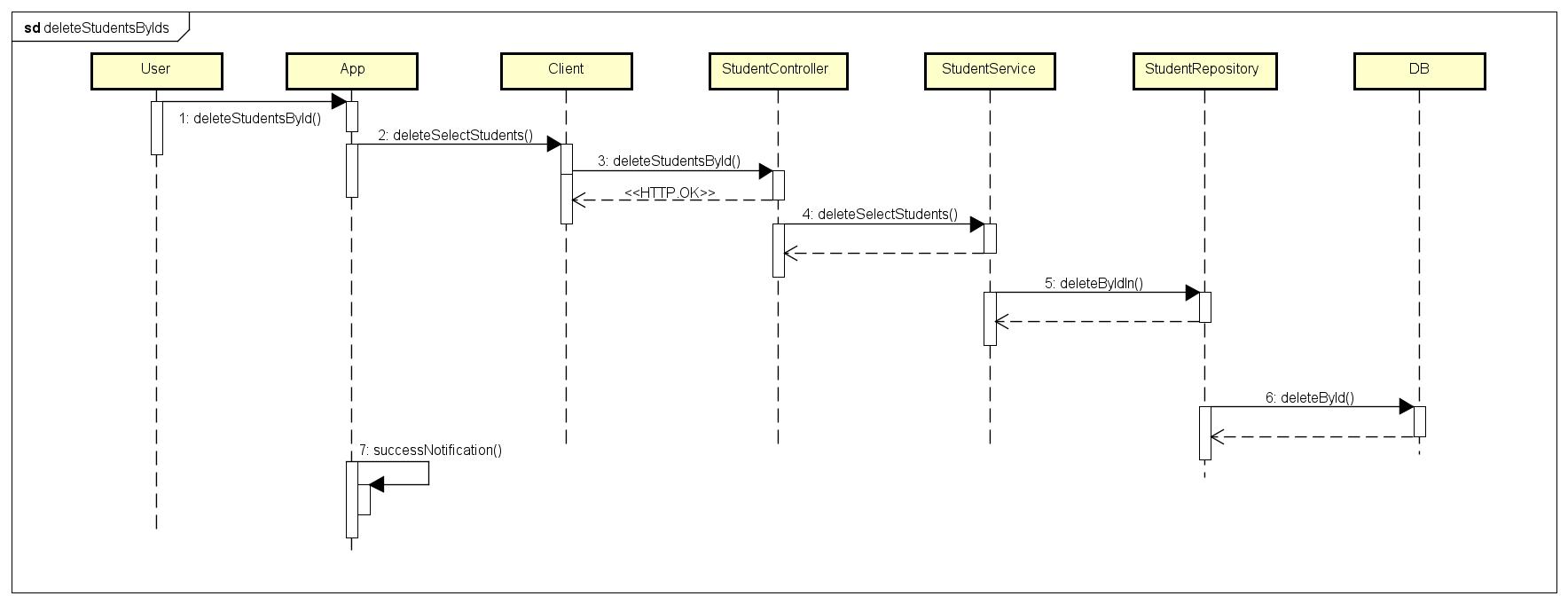


# **4.6 Diagrama de secuencia - deleteStudentsByIds**

**Nombre:** deleteStudentsByIds

**Descripción:** Muestra la secuencia desde que el cliente(frontend) selecciona eliminar a varios estudiantes, se hace la petición Delete al sistema BackEnd, se eliminan de la base de datos, se lanza un status HTTP 200 y se despliega una notificación de éxito.

**Diagrama:**



# **5. Descripción de caso de uso**

https://youtu.be/ArraldzLrmU