DCC

DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2143 - INGENIERÍA DE SOFTWARE

2023 - 1º SEMESTRE

CLASE PRÁCTICA 2:

- Rails, Web y Postman
- Creando API con Rails
- Operaciones CRUD
- Probando API

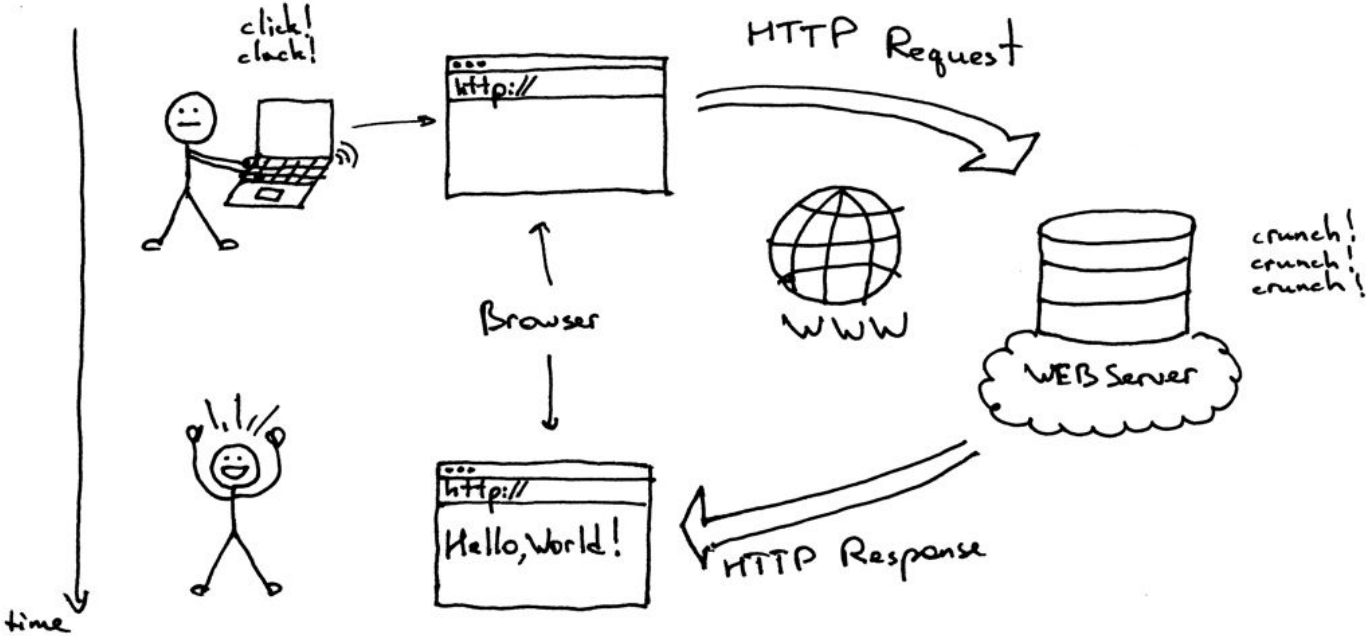
Alison Fernandez Blanco

CLASE PRÁCTICA 2:

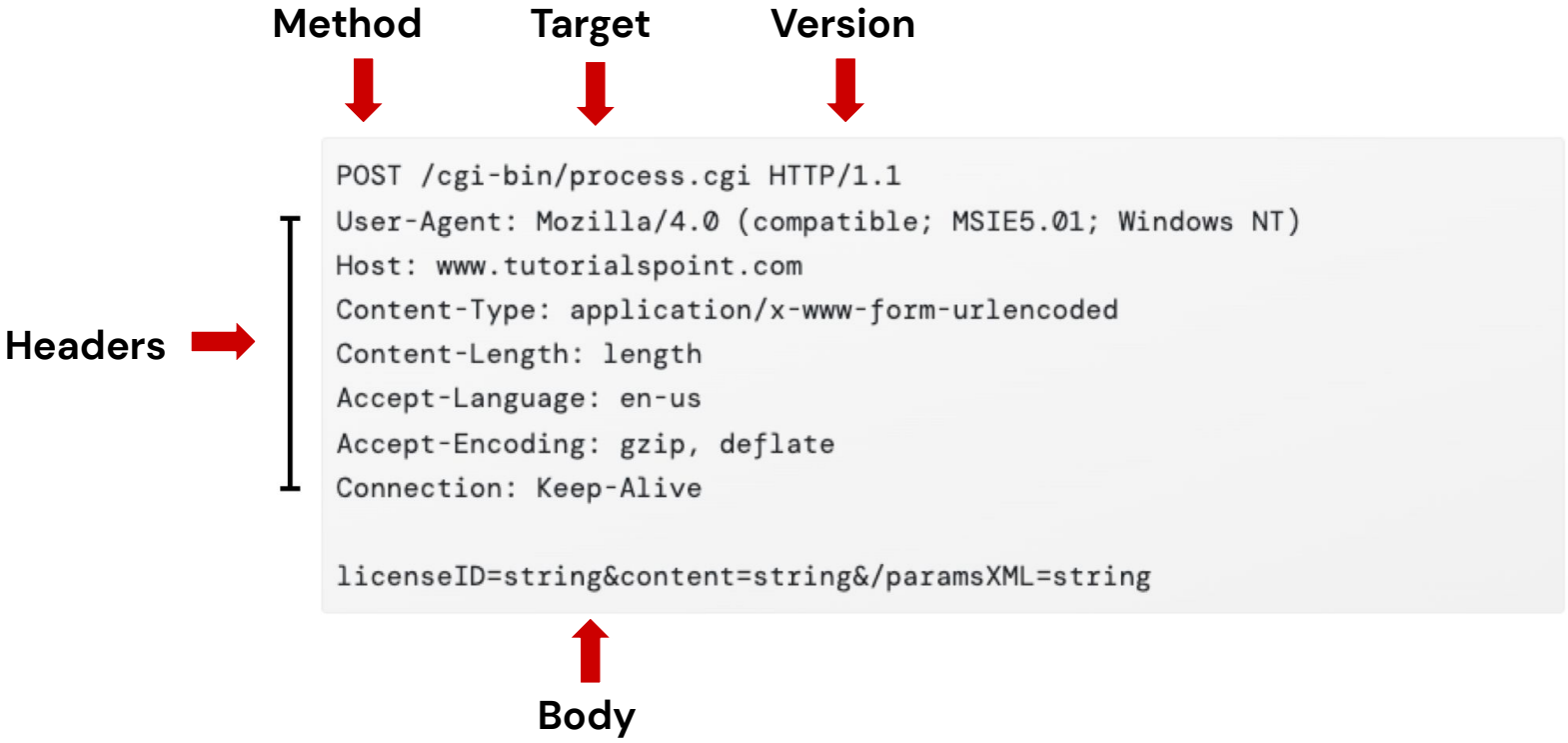
- Rails, Web y Postman
- Creando API con Rails
- Operaciones CRUD
- Probando API

3 ¿Qué es Ruby on Rails?

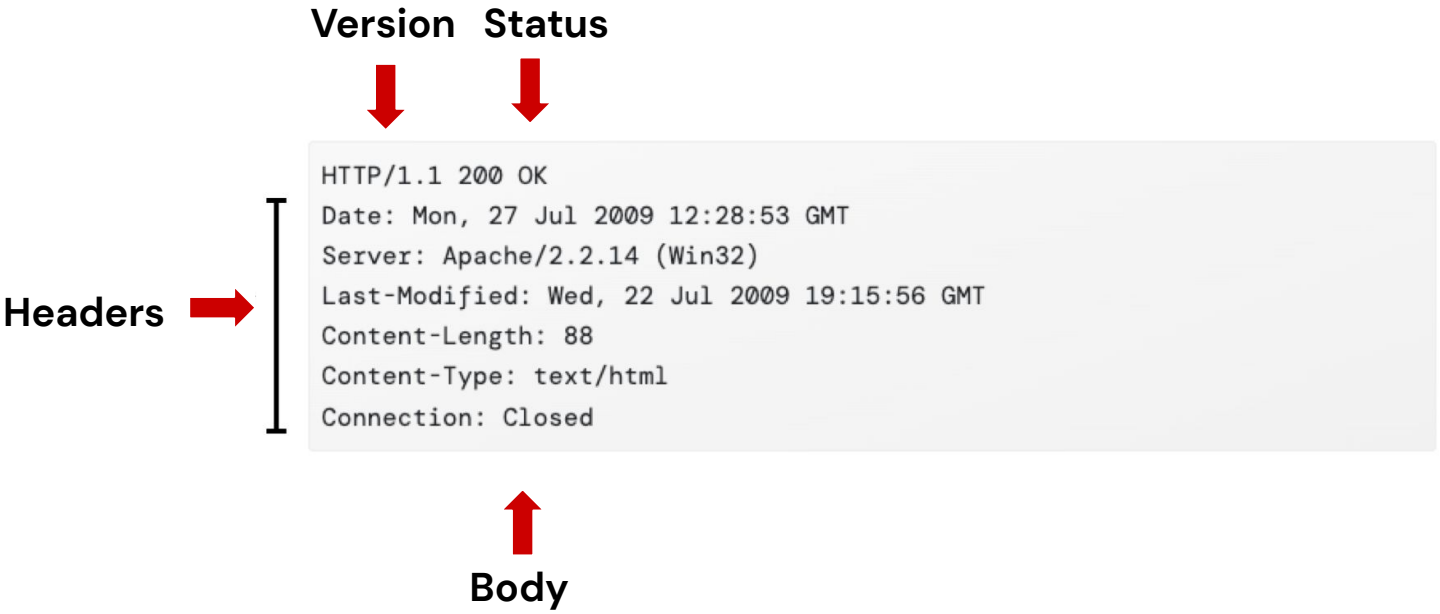
Es un framework de desarrollo de **aplicaciones web** escrito en Ruby.



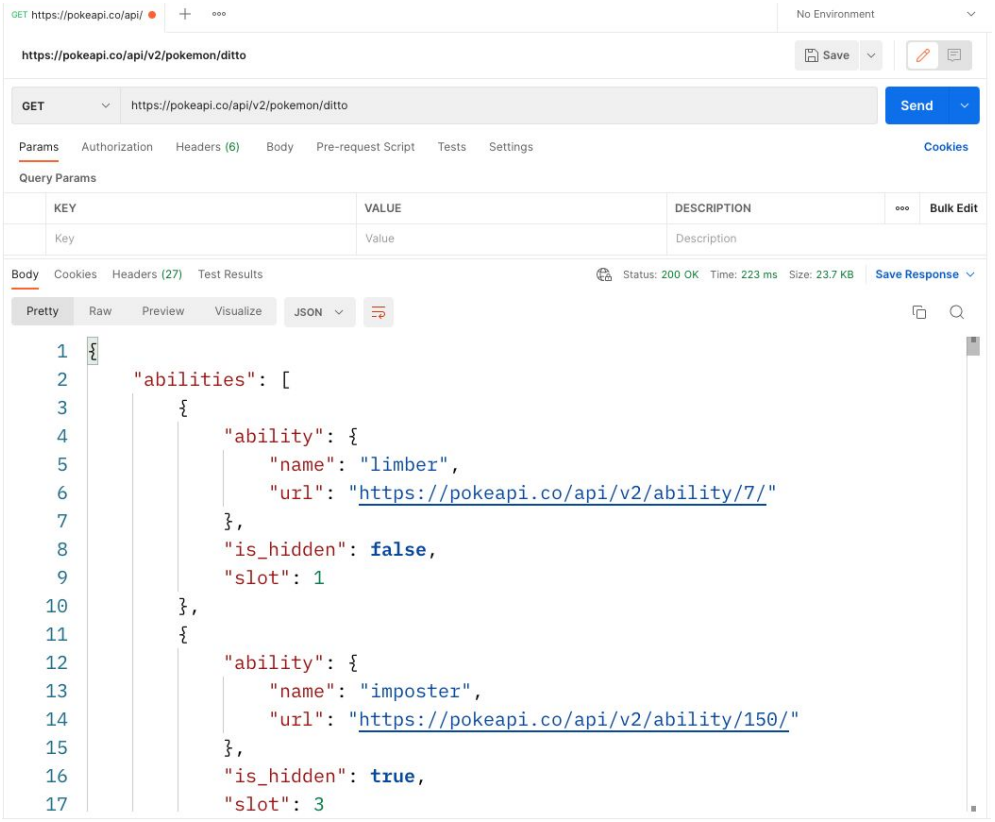
4 HTTP Request



5 HTTP Response



6 Explorando Poke API con Postman



HTTP Request

HTTP Response

7 Instalando Ruby on Rails

Set-up de ayudantia: https://github.com/IIC2143/Setup-Guides

Prerrequisitos:

- Ruby (versión adecuada según la versión de Rails)
- Sqlite3

Para verificar que Rails está instalada correctamente, abrir la consola o terminal y ejecutar el comando:

```
rails --version
```

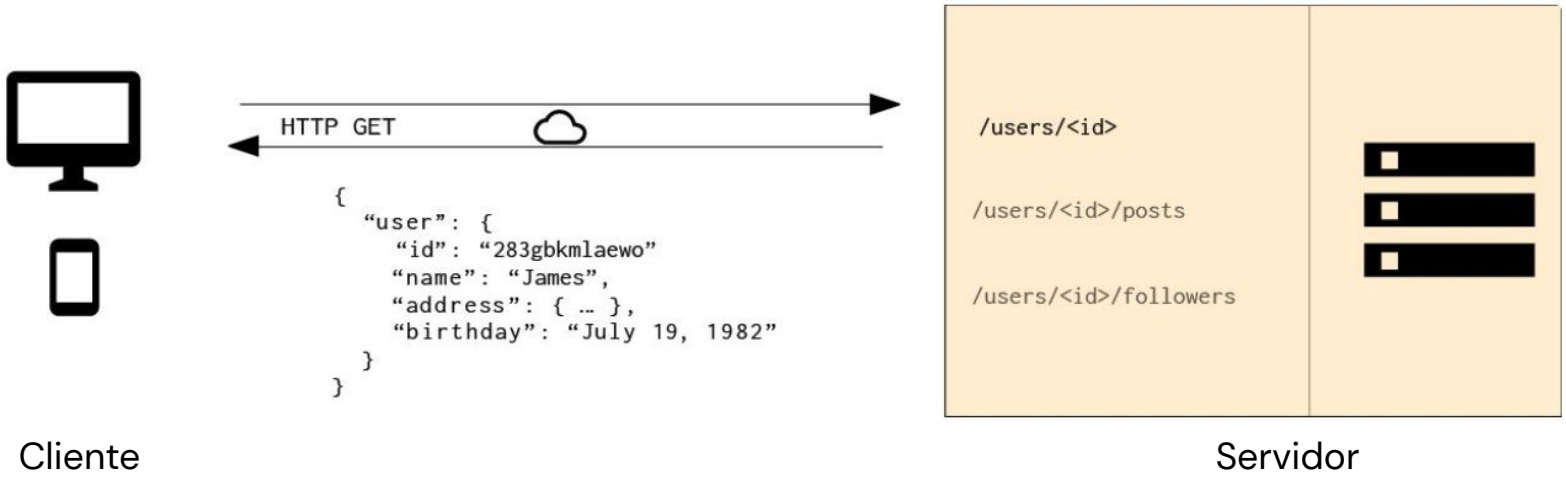
CLASE PRÁCTICA 2:

- Rails, Web y Postman
- **Creando API con Rails**
- Operaciones CRUD
- Probando API

9 REST API

Conjunto de mecanismos que permiten que los clientes accedan a datos del servidor web mediante funciones (GET, PUT, DELETE, etc).

En este caso, las solicitudes de los clientes son similares a las URL que se escriben en el navegador para visitar un sitio web. La respuesta del servidor son datos simples.



10 Creando API

Los siguientes comandos serán realizados en la terminal/consola.

```
rails new my_students --api
```

La anterior línea de comando crea una aplicación API Rails llamada my_students. También instalará las dependencias de gemas que se mencionan en Gemfile usando Bundle install.

```
rails new --help
```

La anterior línea de comando muestra todas las opciones que el generador de aplicación Rails acepta.

```
cd my_students
```

La anterior línea de comando accede al folder my_students.

11 Iniciar el servidor

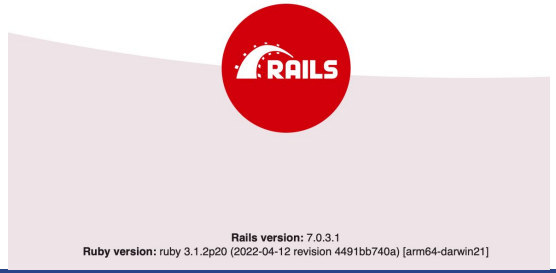
Los siguientes comandos serán realizados en la terminal/consola.

```
bin/rails server
```

La anterior línea de comando inicializa el servidor.

Abre un navegador y accede a:

```
http://localhost:3000
```



12 API my_students

Permitirá al cliente agregar estudiantes, mostrar estudiantes, borrar estudiantes y actualizar los datos. Para ello:

- Definir las rutas para las operaciones
- Crear un controlador Students
- Crear un modelo Student y migrar
- Explorar operaciones y realizar cambios en el controlador Students
- Realizar pruebas con Postman

13 Definiendo rutas

Abre el archivo **config/routes.rb** y realiza los siguientes cambios:

```
Rails.application.routes.draw do
  # Define your application routes per the DSL in
https://guides.rubyonrails.org/routing.html

  # Defines the root path route ("/")
  #root "students#index"
  get "/students", to: "students#index"
  post "/students", to: "students#create"
  get "/students/:id", to: "students#show"
  patch "/students/:id", to: "students#update"
  delete "/students/:id", to: "students#destroy"

end
```

14 Inspeccionando mapeo de rutas

Los siguientes comandos serán realizados en la terminal/consola.

```
bin/rails routes
```

La anterior línea de comando lista las rutas que existen y como están mapeadas. El resultado muestra:

```
Prefix Verb   URI     Pattern                       Controller#Action
    students  GET     /students(.:format)             students#index
              POST    /students(.:format)             students#create
              GET     /students/:id(.:format)         students#show
              PATCH   /students/:id(.:format)         students#update
              DELETE  /students/:id(.:format)         students#destroy
```

15 Creando el controlador Students

Los siguientes comandos serán realizados en la terminal/consola.

```
bin/rails generate controller Students
```

La anterior línea de comando usa el generador de controladores para crear el controlador Student y sus archivos relacionados:

```
create app/controllers/students_controller.rb
invoke test_unit
create test/controllers/students_controller_test.rb
```

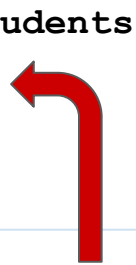
16 Creando modelo Student

Los siguientes comandos serán realizados en la terminal/consola.

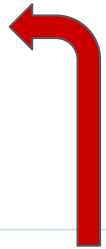
```
bin/rails generate model Student name:text score:integer
```

La anterior línea de comando usa el generador de modelos para crear un modelo Student con atributos name y score. Como resultado se crean los archivos respectivos:

```
invoke  active_record
create    db/migrate/20230313005352_create_students.rb
create    app/models/student.rb
invoke    test_unit
create      test/models/student_test.rb
create      test/fixtures/students.yml
```



Archivo del modelo



Archivo de migración

¿Qué es una migración?

Las migraciones permiten cambiar el esquema de su base de datos, se escriben usando el DSL de Ruby para describir los cambios en sus tablas.

Pueden pensar en cada migración como una nueva 'versión' de la base de datos.

En este caso, **create_table** específica como se construye la tabla **Students**:

- Columna **id**, agregada por defecto, se autorellena (empieza en 1).
- Columnas **name, score** (agregadas por definición)
- Columnas **created_at**, **updated_at** (debido al t.timestamps).

```
class CreateStudents <
ActiveRecord::Migration[7.0]
  def change
    create_table :students do |t|
      t.text :name
      t.integer :score
      t.timestamps
    end
  end
end
```

18 Migrando

Los siguientes comandos serán realizados en la terminal/consola.

```
bin/rails db:migrate
```

La anterior línea de comando corre la migración. En este caso, crea la tabla students.

```
== 20230313005352 CreateStudents: migrating
================================== -- create_table(:students)
-> 0.0015s
== 20230313005352 CreateStudents: migrated (0.0015s)
===========================
```

CLASE PRÁCTICA 2:

- Rails, Web y Postman
- Creando API con Rails
- **Operaciones CRUD**
- Probando API

Interactuando con la base de datos

Operaciones:

- Crear objetos y guardarlos en la base de datos
- Leer
- Actualizar
- Eliminar

21 Crear objetos

Los siguientes comandos serán realizados en la terminal/consola.

```
bin/rails console
> student1=Student.new(name:"Cinthia Sanchez", score:90)
> student1.save
> student1
```

bin/rails console abre una terminal que permite ejecutar código Ruby. En este caso, creamos student1 con **new** y lo guardamos en la base de datos con **save**.

```
> student2=Student.create(name:"Cristobal Campusano", score:85)
> student2
```

Creamos student2 y lo guardamos en la base de datos con **create** (no hay necesidad de llamar a save).

22 Implementando create en el controlador

Abre el archivo del controlador **app/controllers/students_controller.rb** y agregar a la clase las siguientes líneas:

```
# POST /students
def create
  @student = Student.new(student_params)
  if @student.save
    render json: @student
  else
    render json: @student.errors, status: :unprocessable_entity
  end
end
# Agregar al final de la clase por el control de accesos
private
    def student_params
      params.require(:student).permit(:name, :score)
end
```

23 Leer

Los siguientes comandos serán realizados en la terminal/consola con bin/rails console.

```
> Student.all
```

Retorna una colección con todos los estudiantes de la base de datos.

```
> Student.first
```

Retorna el primer estudiante (id=1).

```
> Student.find_by(name:"Cinthia Sanchez")
```

Retorna el primer estudiante que tenga como nombre Cinthia Sanchez.

```
> Student.where(score:85).order(created_at: :desc)
```

Busca todos los estudiantes con score igual a 80 y los ordena descendentemente por created_at.

24 Implementando index y show en el controlador

Abre el archivo del controlador **app/controllers/students_controller.rb** y agregar a la clase las siguientes líneas:

```ruby
# GET /students
def index
  @students = Student.all
  render json: @students
end

# GET /students/1
def show
  @student = Student.find(params[:id])
  render json: @student
end
```

25 Actualizar

Los siguientes comandos serán realizados en la terminal/consola con bin/rails console.

```
> student = Student.find_by(name:"Cinthia Sanchez")
> student.score = 98
> student.save
```

Busca el estudiante de nombre Cinthia Sanchez, actualiza su nota a 98 y lo guarda en la base de datos.

```
> student = Student.find_by(name:"Cristobal Campusano")
> student.update(score: 90)
```

Busca el estudiante de nombre Cristobal Campusano, cambia su score a 90 y actualiza la base de datos.

26 Implementando update en el controlador

Abre el archivo del controlador **app/controllers/students_controller.rb** y agregar a la clase las siguientes líneas:

```ruby
# PATCH /students/1
def update
  @student = Student.find(params[:id])
  if @student.update(student_params)
      render json: @student
  else
      render json: @student.errors, status: :unprocessable_entity
  end
end
```

27 Eliminar

Los siguientes comandos serán realizados en la terminal/consola con bin/rails console.

```
> student = Student.find_by(name:"Cristobal Campusano")
> student.destroy
```

Busca al estudiante con nombre Cristobal Campusano y lo borra de la base de datos. Para borrar una colección de estudiantes usa **destroy_all**.

28 Implementando destroy en el controlador

Abre el archivo del controlador **app/controllers/students_controller.rb** y agregar a la clase las siguientes líneas:

```ruby
# DELETE /students/1
def destroy
  @student = Student.find(params[:id])
  @student.destroy
end
```

CLASE PRÁCTICA 2:

- Rails, Web y Postman
- Creando API con Rails
- Operaciones CRUD
- **Probando API**

30 Creando Request en Postman para probar API

1. Abrir Postman, ir a Create new y seleccionar la opción “HTTP Request”.
2. En la vista principal seleccione el tipo de request y agrega la URL correspondiente como ser: localhost:3000/articles
3. Presione el botón Send y observe el resultado en JSON.
4. Presione el botón Save y guarde su request.

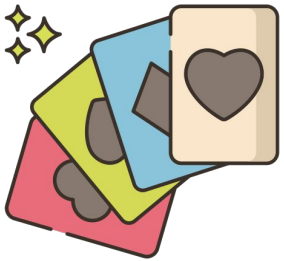
31 Importando requests en Postman para probar API

1. Descargar *RequestsAPIMyStudents.postman_collection.json* (https://www.dropbox.com/sh/ri886rhknjqdfut/AADYDDjsM_0FAzmwLeBpzU7Wa?dl=0).
2. Abrir Postman y seleccionar un workspace.
3. Seleccionar 'Import' y elegir *RequestsAPIMyStudents.postman_collection.json*.
4. Desplegar las requests en el menú y ejecutarlas en orden.

Ejercicio por décimas extra

- Agregue el siguiente método al controlador *Student*:
  - filter - método que retorna en formato JSON los datos de los estudiantes que tienen una nota igual al argumento del request (params[:score]).
- Agregue la ruta correspondiente ('/students/filter/:score') al archivo de rutas.
- Crea la request en Postman para demostrar que el método filter funciona correctamente.
- Entrega un solo archivo pdf hasta el 15 de Marzo a las 11:30 AM (https://forms.gle/Ze8ZNREmF1bo9oz57). El archivo debe incluir:
  - El código del controlador implementado (students_controller.rb)
  - El código del archivo de rutas (routes.rb).
  - Un screenshot de Postman donde se muestre que el API funciona.

Nota: Los que entreguen este ejercicio tendrán una décima extra para la I1!



33 Material adicional

- Documentación de Ruby on Rails: https://guides.rubyonrails.org/getting_started.html
- Documentación Active Record: https://guides.rubyonrails.org/active_record_basics.html
- Documentación migraciones: https://guides.rubyonrails.org/active_record_migrations.html
- Documentación validaciones: https://guides.rubyonrails.org/active_record_validations.html
- Material temporal del curso: https://github.com/Balison/IIC2143-IngenieriaSoftware

¿Consultas?