

Tutorial 3 - Parte 3

Spring

FCHAVEZ

Inyección de dependencias

- Las clases son un mecanismo de modularización que nos permite hacer administrable la complejidad de los proyectos.
- Las clases, que modelan funciones sencillas, deben trabajar unas con otras para crear comportamiento complejo, lo que crea dependencias.
- El concepto de inyección de dependencias señala que una clase B que necesita de una clase A NO debe instanciar los objetos de tipo A que necesita, sino recibirlas (mediante un método set) para evitar dependencias entre la implementación de A y la de B.

Inversión de control

- La inyección de dependencias potencia la inversión de control.
 - En la inyección de control un “agente” es el encargado de injectar las dependencias de forma adecuada. Para el caso de Spring este agente es el Spring Framework.
 - El programador cuenta con la anotación `@Autowired` para indicar a Spring Framework donde injectar dependencias.

IoC en el proyecto

```
@Repository  
public class UsuarioRepository implements UserRepository{  
    private UsuarioCrudRepository repository;  
    private UserMapper mapper;
```

...



Dependencias

Nota: en este momento ninguna de estas dos variables es instancia, por lo que si las ocuparemos obtendríamos un NullPointerException.

IoC en el proyecto

```
@Repository  
public class UsuarioRepository implements UserRepository{  
    @Autowired  
    private UsuarioCrudRepository repository;  
    @Autowired  
    private UserMapper mapper;
```

Solicitamos a Spring Framework que instancie estos objetos y que ademas los inyecte a la instancia de UsuarioRepository que también creará.

Wow

¡Qué mágico!

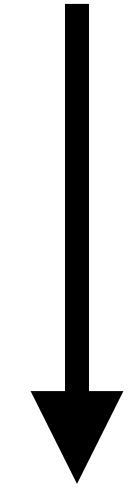
Buenoooo ... no tanto.

Sólo puede instanciar e injectar objetos que sean componentes de Spring

¿Dudas?

- ¿Y dónde veo que las dependencias que estamos inyectando son componentes de Spring?

```
public interface UsuarioCrudRepository extends CrudRepository<Usuario, Integer> {  
    Usuario findByLogin(String login);  
}
```



Al heredar de CrudRepository (como esta clase es un componente de Spring) UsuarioCrudRepository se hace un componente

¿Dudas?

- ¿Y dónde veo que las dependencias que estamos inyectando son componentes de Spring?

```
@Mapper(componentModel = "spring")
public interface UserMapper {
    @Mappings({
        @Mapping(source = "id", target = "id"),
        @Mapping(source = "login", target = "login"),
        @Mapping(source="contrasena", target="password"),
        @Mapping(source="correo", target="email")
    })
}
```

Le decimos que use el modelo de componentes de Spring

Terminamos ...

- Por fin terminamos el repositorio de nuestra aplicación

Terminamos ...

- Descansa un poco



porque vamos a seguir

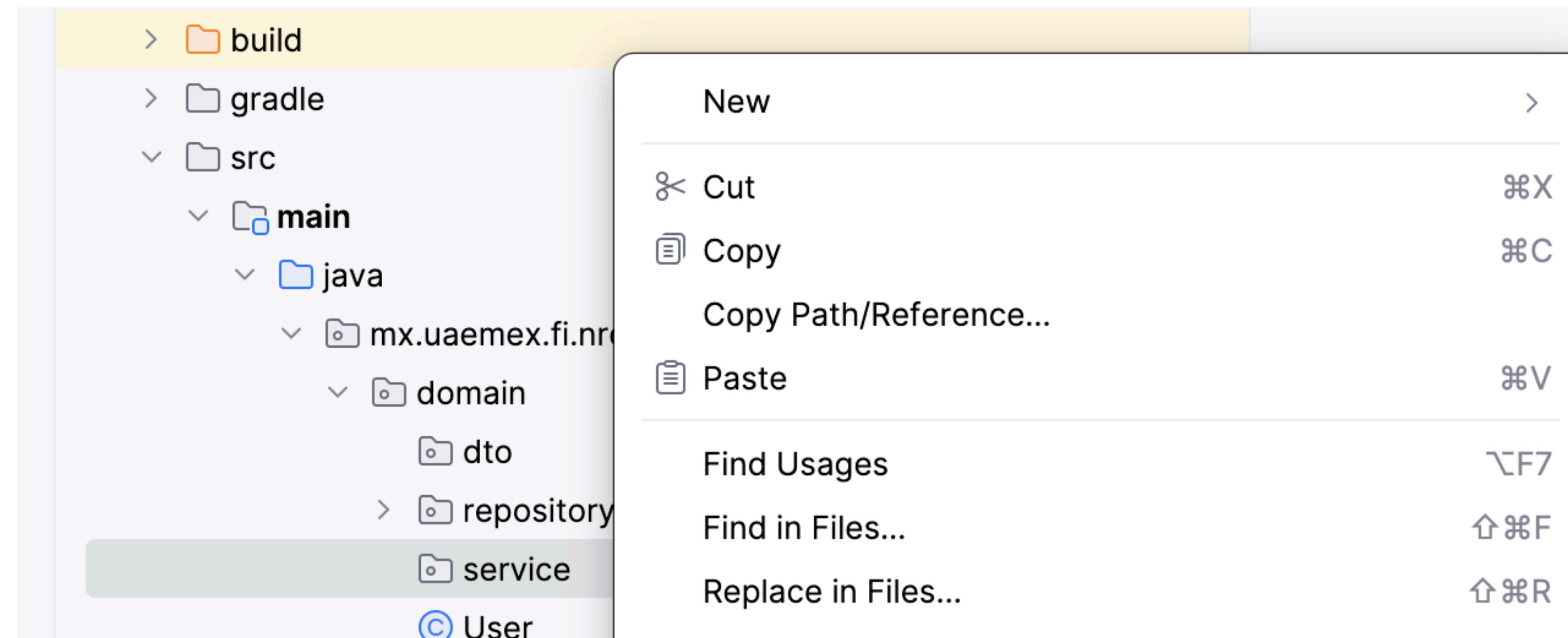
Servicios de dominio

- Interfaz entre el controlador y el repositorio



Servicio de Usuarios

- Click derecho sobre el paquete: service



Servicio de Usuarios

New Java Class

© UsuarioService

© Class

Interface

Record

Enum

Annotation

Exception

Servicio de Usuarios

```
@Service  
public class UsuarioService {  
}
```

Hacemos que la clase sea un servicio

Servicio de Usuarios

```
@Service
public class UsuarioService {

    @Autowired
    private UsuarioRepository usuarioRepository;

    public List<User> findAll() {
        return this.usuarioRepository.getAll();
    }

    public Optional<User> getUser(int id){
        return this.usuarioRepository.getUser(id);
    }

    public User save(User user) {
        return this.usuarioRepository.save(user);
    }

    public boolean delete(int id){
        return getUser(id).map(user -> {usuarioRepository.deleteById(id); return true;}).orElse(false);
    }
}
```

Servicio

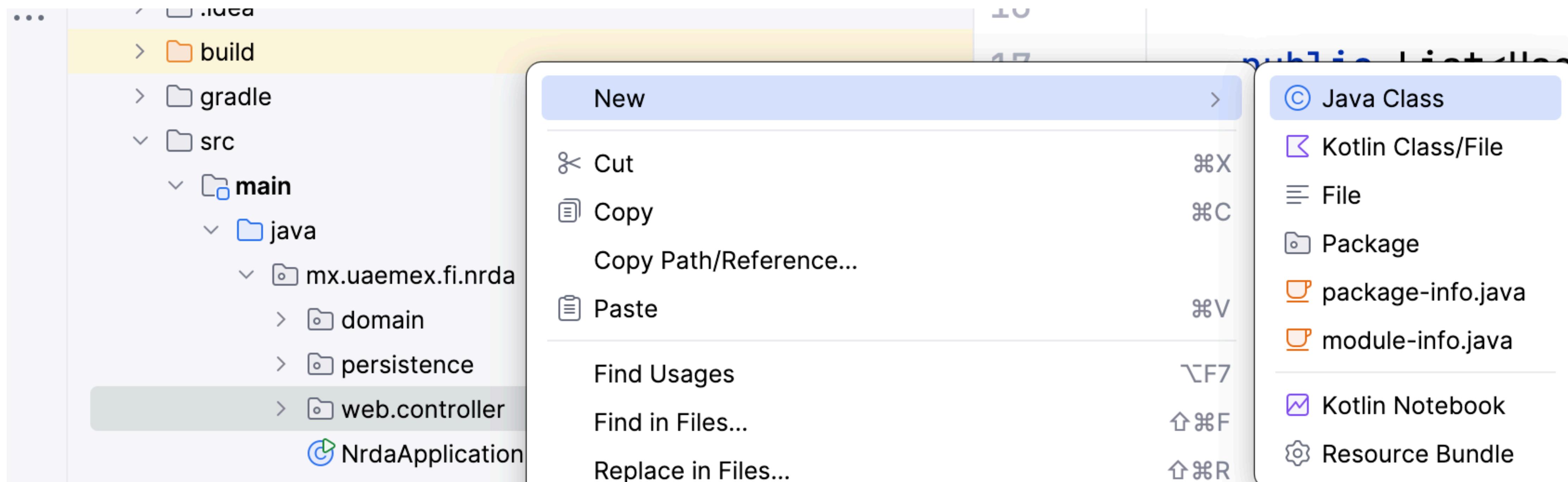


Controll time

- Ya estamos listos para hacer el controlador

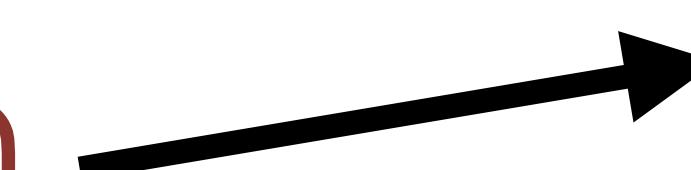


Controller



UsuarioController

```
@RestController  
@RequestMapping("/users")  
public class UserController {  
    @Autowired  
    private UsuarioService usuarioService;  
  
    @GetMapping("all")  
    public List<User> getAll(){  
        return usuarioService.findAll();  
    }  
}
```

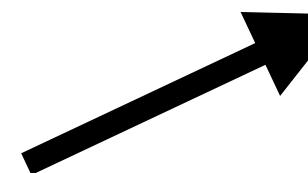
 Lo hacemos un controlador

UsuarioController

```
@RestController
@RequestMapping( "/users" )
public class UserController {
    @Autowired
    private UsuarioService usuarioService;

    @GetMapping( "all" )
    public List<User> getAll(){
        return usuarioService.findAll();
    }
}
```

url de este controlador

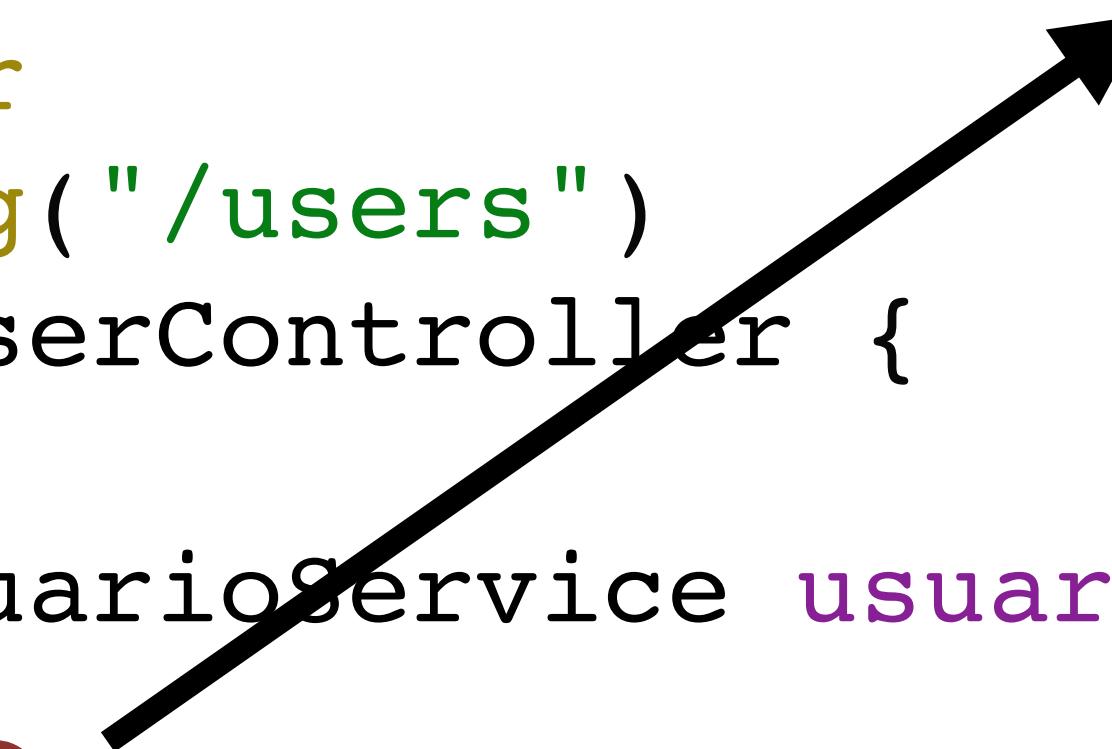


UsuarioController

```
@RestController
@RequestMapping( "/users" )
public class UserController {
    @Autowired
    private UsuarioService usuarioService;

    @GetMapping( "all" )
    public List<User> getAll(){
        return usuarioService.findAll();
    }
}
```

Verbo http al que responde

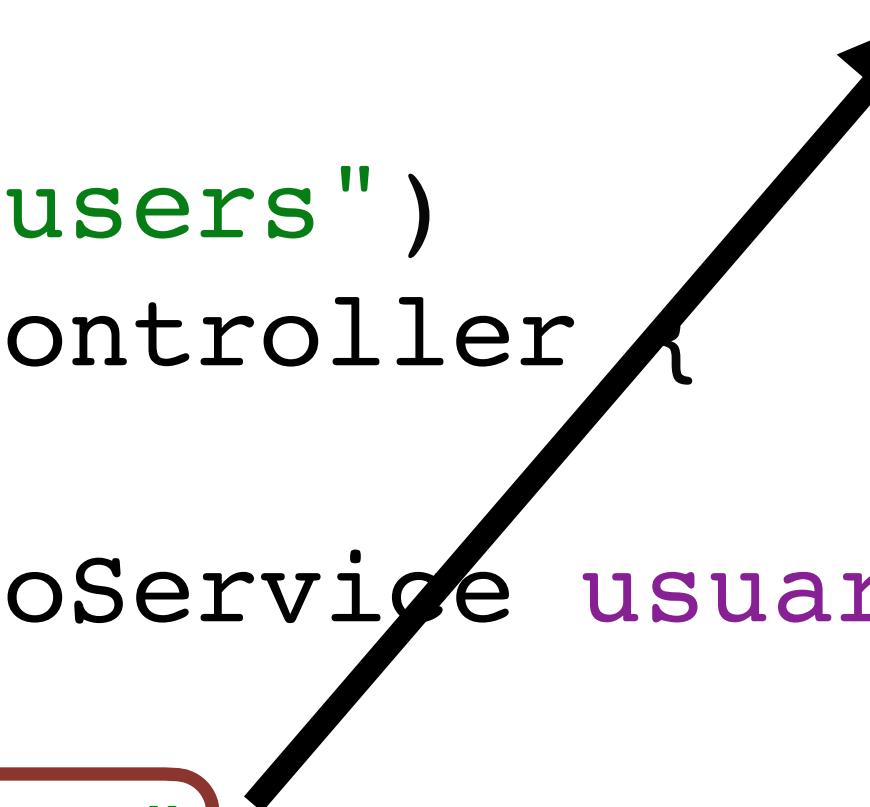


UsuarioController

```
@RestController
@RequestMapping( "/users" )
public class UserController {
    @Autowired
    private UsuarioService usuarioService;

    @GetMapping( "all" )
    public List<User> getAll(){
        return usuarioService.findAll();
    }
}
```

url de la consulta



UsuarioController

Verbo http y url

```
@GetMapping("/{id}") no usages
public Optional<User> getUser(int id){
    return usuarioService.getUser(id);
}

@PostMapping("/save") no usages
public User save(User user){
    return usuarioService.save(user);
}

@DeleteMapping("/{id}") no usages
public boolean delete(int userId){
    return usuarioService.delete(userId);
}
```

UsuarioController

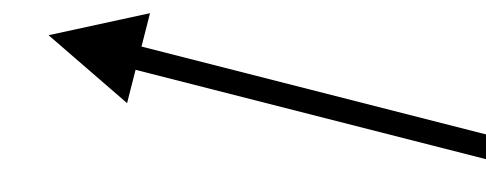
```
    @GetMapping("/{id}") no usages
    public Optional<User> getUser(int id){
        return usuarioService.getUser(id);
    }

    Verbo http y url
    ↗ @PostMapping("/save") no usages
    public User save(User user){
        return usuarioService.save(user);
    }

    @DeleteMapping("/{id}") no usages
    public boolean delete(int userId){
        return usuarioService.delete(userId);
    }
```

UsuarioController

Verbo http y url



```
@GetMapping("/{id}") no usages
public Optional<User> getUser(int id){
    return usuarioService.getUser(id);
}

@PostMapping("/save") no usages
public User save(User user){
    return usuarioService.save(user);
}

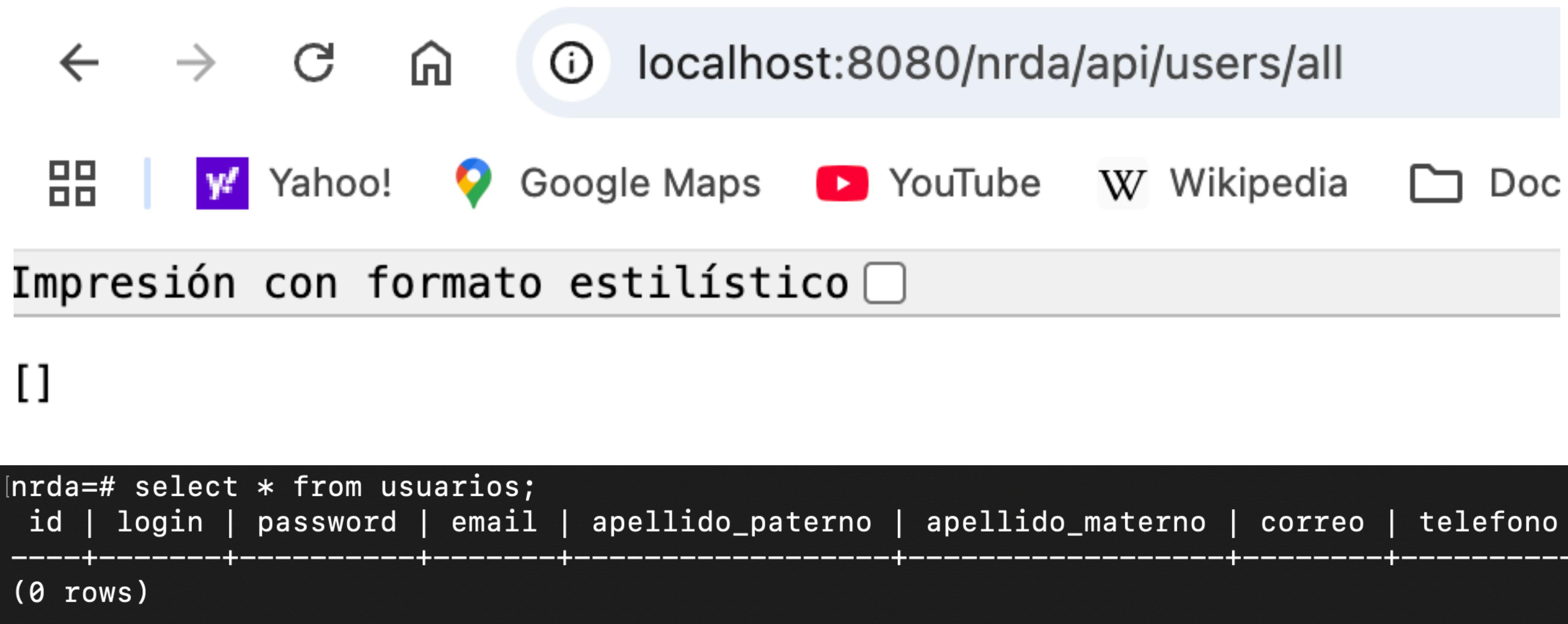
@DeleteMapping("/{id}") no usages
public boolean delete(int userId){
    return usuarioService.delete(userId);
}
```

¡Listo!

- Hora de probar



Prueba



The screenshot shows a web browser interface with a terminal-like query results display. The browser's address bar shows the URL `localhost:8080/nrda/api/users/all`. Below the address bar, there are several browser navigation icons and links to external sites like Yahoo!, Google Maps, YouTube, Wikipedia, and Doc. The main content area displays a terminal-style output:

```
[nrda=# select * from usuarios;
 id | login | password | email | apellido_paterno | apellido_materno | correo | telefono
----+-----+-----+-----+-----+-----+-----+-----+
 (0 rows)
```

Prueba

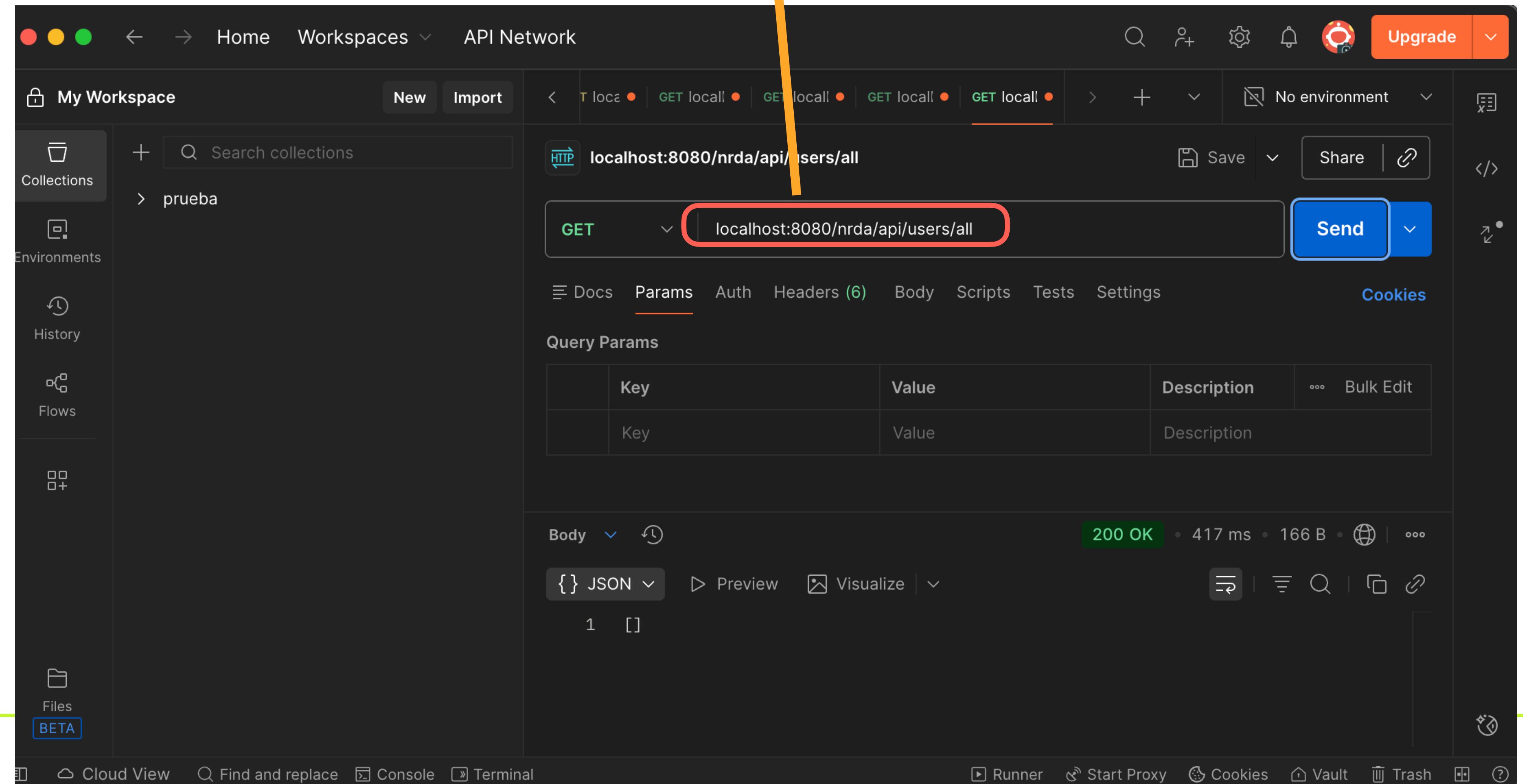


POSTMAN

Prueba

Postman

url



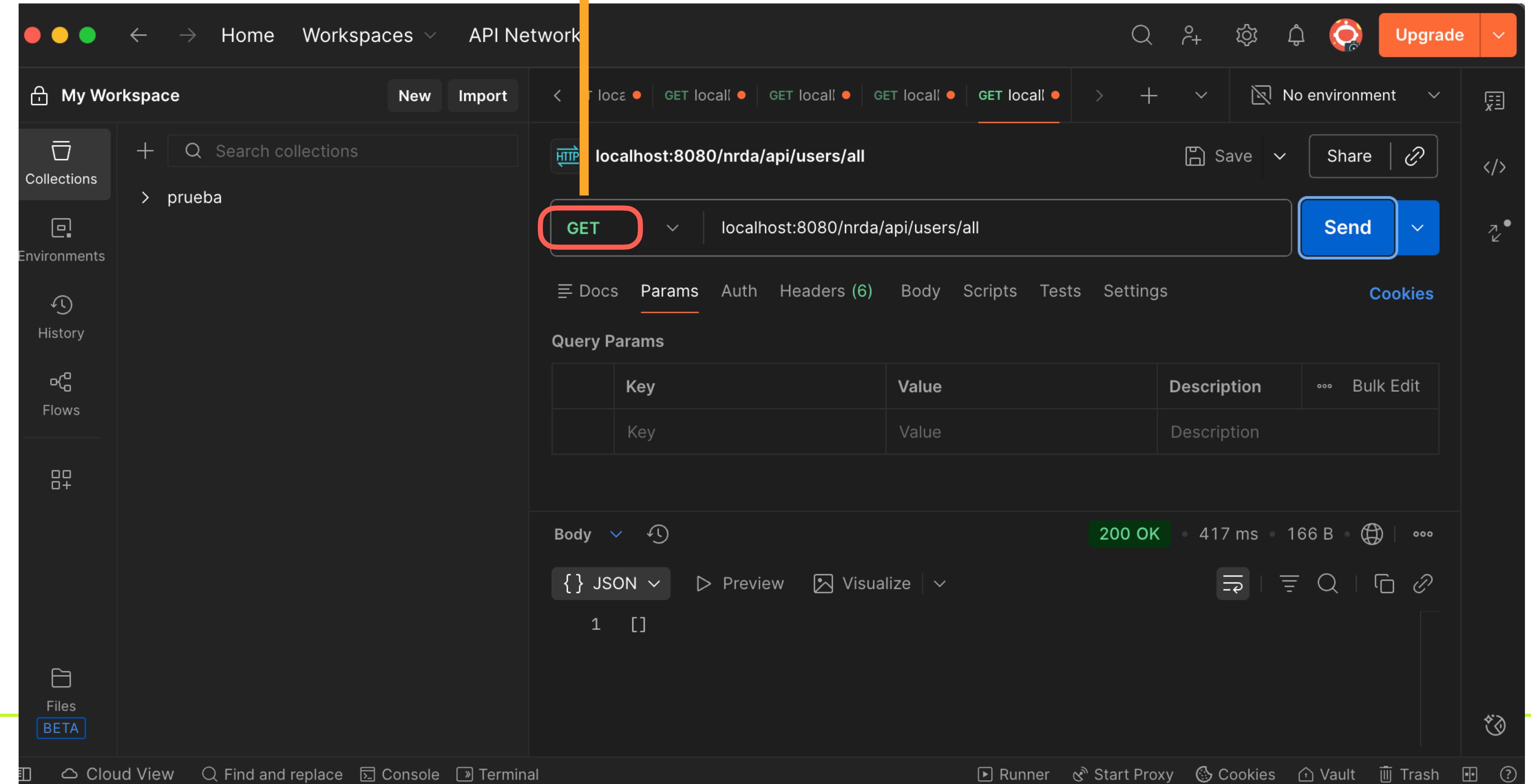
The screenshot shows the Postman application interface. On the left, there's a sidebar with icons for Collections, Environments, History, Flows, and Files (BETA). The main workspace is titled "My Workspace" and contains a collection named "prueba". A search bar for collections is present. In the center, a request is being built for the URL `localhost:8080/nrda/api/users/all`. The method is set to `GET`. The "Params" tab is selected, showing a table for "Query Params" with columns for Key, Value, Description, and Bulk Edit. The table is currently empty. Below the table, the "Body" tab is selected, showing a JSON response with a single object: `1 []`. The response status is `200 OK`. At the bottom, there are buttons for Cloud View, Find and replace, Console, Terminal, Runner, Start Proxy, Cookies, Vault, Trash, and Help.

FCHAVEZ

Prueba

Verbo http

Postman



Postman interface showing a collection named "prueba". A GET request is selected with the URL `localhost:8080/nrda/api/users/all`. The "GET" button is highlighted with a red box. An orange arrow points from the text "Verbo http" to this button. The "Params" tab is selected in the request details panel, which is currently empty. The response status is 200 OK, with a response time of 417 ms and a response size of 166 B.

HTTP `localhost:8080/nrda/api/users/all`

GET `localhost:8080/nrda/api/users/all`

Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body

200 OK • 417 ms • 166 B

JSON Preview Visualize

1 []

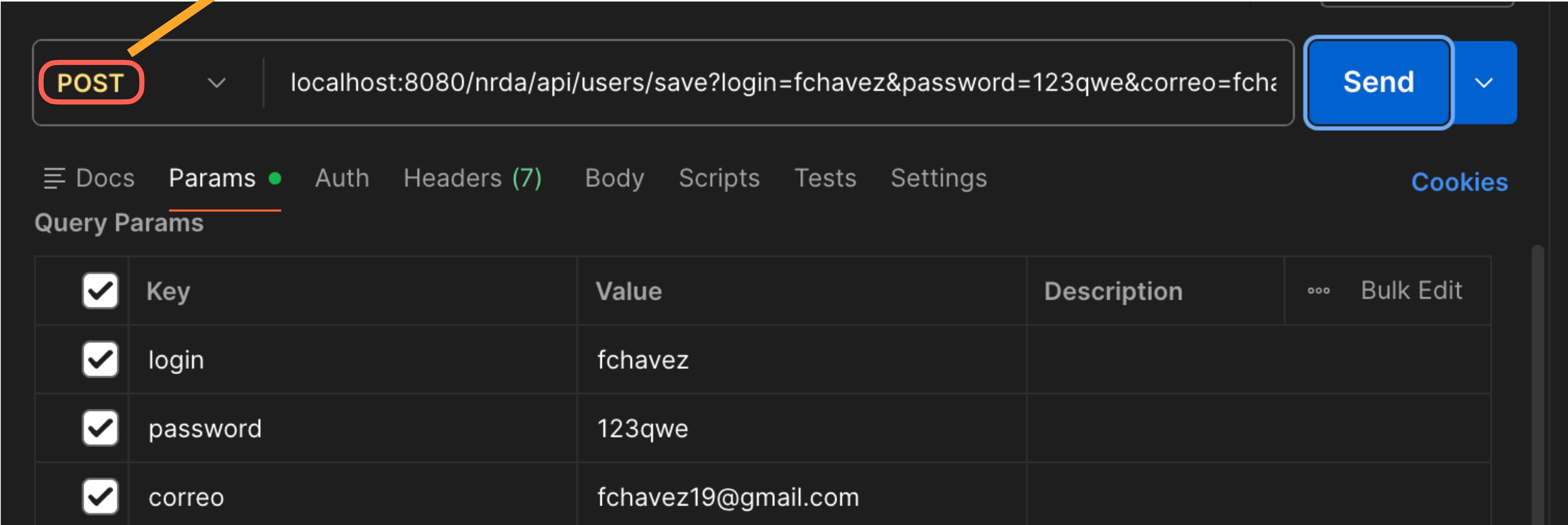
Cloud View Find and replace Console Terminal Runner Start Proxy Cookies Vault Trash

FCHAVEZ

Prueba

Postman

Verbo http



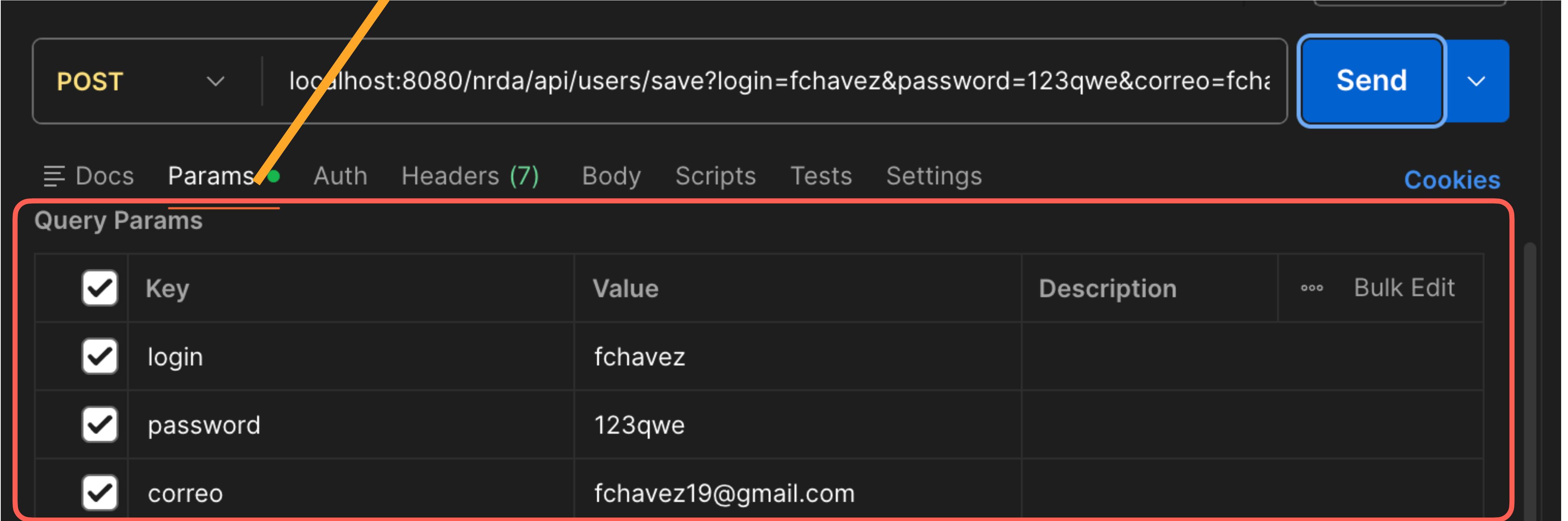
The screenshot shows the Postman interface with a POST request setup. The 'POST' button is highlighted with a red border. The URL is set to `localhost:8080/nrda/api/users/save?login=fchavez&password=123qwe&correo=fchavez19@gmail.com`. The 'Send' button is visible on the right. Below the URL, the 'Params' tab is selected, showing the following query parameters:

Key	Value	Description	...	Bulk Edit
login	fchavez			
password	123qwe			
correo	fchavez19@gmail.com			

Prueba

Parámetros

Postman



POST localhost:8080/nrda/api/users/save?login=fchavez&password=123qwe&correo=fchavez19@gmail.com Send

Params

Query Params

Key	Value	Description	...	Bulk Edit
login	fchavez			
password	123qwe			
correo	fchavez19@gmail.com			

FCHAVEZ