Desarrollo Backend con NodeJS

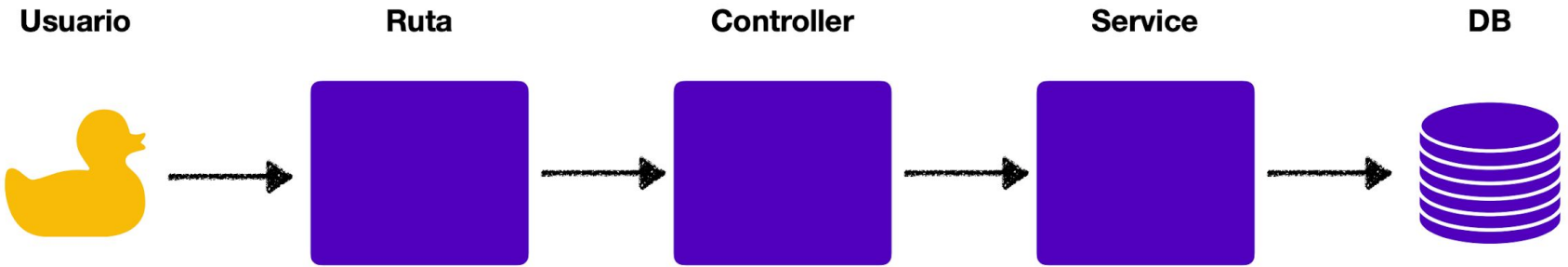
Validaciones y paginado

course[It]

Validación y manejo de errores

Validación

Es el acto de confirmar que los datos que necesitamos **están llegando (a través de la request)** correctamente a nuestra api. Al ser lógica de negocio las validaciones se realizan **en el controller.**



Sin validación

```
  async createUser(req, res) {
    const data = req.body;
    const user = await this.userService.addUser(data);
    res.send("usuario creado");
  }
}
```

Con validación

```
  async createUser(req, res) {
    const data = req.body;
    if (data.name && data.age) {
      const user = await this.userService.addUser(data);
      res.send("usuario creado");
    }
  }
}
```

Manejo de errores

Manejo de errores

Es la acción de informar a **quien consume (a través de la response)** nuestra API si las acciones que intentó realizar fueron un éxito o tiene algún error. El manejo de errores también es lógica de negocio y también se realiza en el controller.

Sin manejo de errores

```
  async createUser(req, res) {
    const data = req.body;
    if (data.name && data.age) {
      const user = await this.userService.addUser(data);
      res.send("usuario creado");
    } else {
      res.status(400).send("falta información obligatoria");
    }
  }
}
```

Pero Beeeel, ahí estamos manejando un error

Si, bueno, es cierto, en este caso el error lo podemos detectar en la ruta

```
  async createUser(req, res) {
    const data = req.body;
    if (data.name && data.age) {
      const user = await this.userService.addUser(data);
      res.send("usuario creado");
    } else {
      res.status(400).send("falta información obligatoria");
    }
  }
}
```

¿Pero qué pasa cuando el error viene del service?

```
  async createUser(req, res) {
    const data = req.body;
    if (data.name && data.age) {
      const user = await this.userService.addUser(data);
      res.send("usuario creado");
    } else {
      res.status(400).send("falta información obligatoria");
    }
  }
}
```

Try / Catch

Es un bloque de código condicional que primero intenta realizar una acción, si esa acción devuelve una excepción (error) entonces la “atrapa

```
try {
    nonExistentFunction();
} catch (error) {
    console.error(error);
    // expected output: ReferenceError: nonExistentFunction is not defined
}
```

Manejo de errores con Try / Catch

```
  async createUser(req, res) {
    const data = req.body;

    if (body.name && body.age) {
      try {
        const user = await this.userService.addUser(data);
        res.send("usuario creado");
      } catch (e) {
        console.log(e);
        res.status(500).send("error en la creación");
      }
    } else {
      res.status(400).send("Falta info obligatoria");
    }
  }
}
```

Paginado

Paginado

El paginado es la acción de devolver una lista dividida en partes (páginas), esto nos permite realizar queries más pequeñas y no sobrecargar la base de datos

Método skip

En mongoose, el método skip sirve para saltar resultados a la hora de devolver los registros, lo que es especialmente útil para armar un paginado

```
getUsers(data) {
      // esta query va a traer todos los resultados ignorando los primeros tres
      const query = User.find().skip(3).exec();
      return user
  }
```

Método limit

En mongoose, el método limit sirve para devolver una cantidad máxima de resultados

```
getUsers(data) {
      // esta query va a traer diez resultados ignorando los primeros tres
      const query = User.find().skip(3).limit(10).exec();
      return user
  }
```

Paginado

Para armar un paginado necesitamos un límite por página y un valor de offset (la cantidad que tenemos que ignorar según el número de página en la que estamos)

Para calcular el offset hacemos: limit x (pageNumber - 1) = offset

Por ej, para un límite de veinte entradas:

Página 1: 20 x (1 -1) = 20 x 0 = 0
Página 2: 20 x (2 -1) = 20 x 1 = 20
Página 3: 20 x (3 -1) = 20 x 2 = 40

Todo muy lindo pero... ¿De donde obtenemos esta data?

req.query

Es una propiedades del objeto req que nos permiten acceder a información en la url

- req.query nos da acceso a los parámetros que se envíen a través de la url a la ruta (vienen desde el cliente, no los controlamos desde el servidor)

```
http://localhost:3000/getusers?limit=20&page=2
```