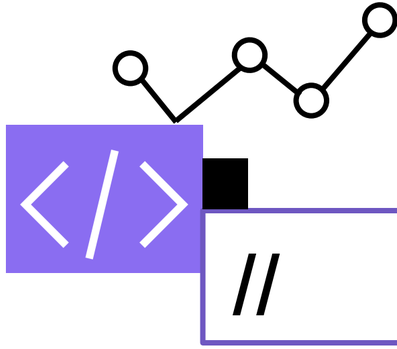
DIGITALHOUSE

PUT, PATCH y DELETE en Go

Certified Tech Developer
The Ultimate Degree

Índice

01 PUT



¿Cómo podemos recibir una petición PUT?

```
{}
pr.POST("/", p.Store())
pr.GET("/", p.GetAll())
pr.PUT("/:id", p.Update())
```

Petición/Respuesta

Al recibir una solicitud **PUT** con un **path parameter** se nos indica el ID del producto almacenado que se está intentando reemplazar.

Petición:

PUT localhost:8080/products/1

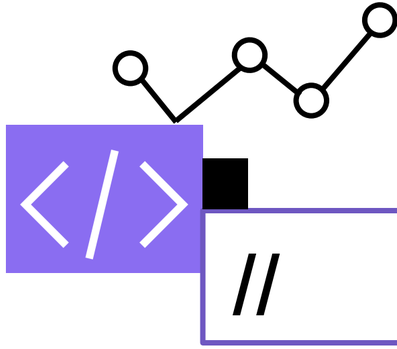
JSON Auth Query Header 2

```
{
  "nombre": "heladera",
  "tipo": "electrodomesticos",
  "cantidad": 2,
  "precio": 40000
}
```

Respuesta:

Preview Header 3 Cookie

```
{
  "id": 1,
  "nombre": "heladera",
  "tipo": "electrodomesticos",
  "cantidad": 2,
  "precio": 40000
}
```



Paso 1 - Agregar método en interface

- Definir un servicio web mediante el método **PUT**, el cual tendrá como path "**products/:id**".
- Agregar los métodos a utilizar en las interfaces de **Repository** y **Service**.

```
{}
type Repository interface {
   GetAll() ([]Product, error)
   Store(id int, name, productType string, count int, price float64) (Product, error)
   LastID() (int, error)
   Update(id int,  name, productType string, count int, price float64) (Product, error)
}
```

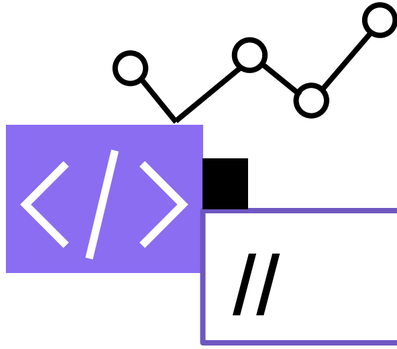
```
{}
type Service interface {
   GetAll() ([]Product, error)
   Store(name, productType string, count int, price float64) (Product, error)
   Update(id int, name, productType string, count int, price float64) (Product, error)
}
```

Paso 2 - Repositorio

Se implementa la funcionalidad para actualizar el producto en memoria en caso de que coincida con el ID enviado. Caso contrario, retorna un error.

```
{}
```

```go
func (r *repository) Update(id int,  name, productType string, count int, price float64)
(Product, error) {
   p := Product{Name: name, Type: productType, Count: count,Price: price}
   updated := false
   for i := range ps {
       if ps[i].ID == id {
           p.ID = id
           ps[i] = p
           updated = true
       }
   }
   if !updated {
       return Product{}, fmt.Errorf("Producto %d no encontrado", id)
   }
   return p, nil
}
```



Paso 3 - Servicio

Dentro del servicio, se llama al repositorio para que proceda a actualizar el producto.

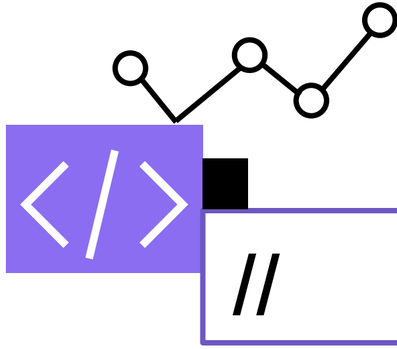
```
{}
func (s *service) Update(id int, name, productType string, count int,
price float64) (Product, error) {

   return s.repository.Update(id, name, productType, count, price)
}
```

Paso 4

Se agrega el controlador **Update** en el **Handler** de productos.

```
func (c *Product) Update() gin.HandlerFunc {
   return func(ctx *gin.Context) {
       token := ctx.GetHeader("token")
       if token != "123456" {
           ctx.JSON(401, gin.H{ "error": "token inválido" })
           return
       }
       id, err := strconv.ParseInt(ctx.Param("id"),10, 64)
       if err != nil {
           ctx.JSON(400, gin.H{ "error":  "invalid ID"})
           return
       }
       var req request
       if err := ctx.ShouldBindJSON(&req); err != nil {
           ctx.JSON(400, gin.H{ "error": err.Error() })
           return
       }
       if req.Name == "" {
           ctx.JSON(400, gin.H{ "error": "El nombre del producto es requerido"})
           return
       }
       if req.Type == "" {
           ctx.JSON(400, gin.H{ "error": "El tipo del producto es requerido"})
           return
       }
       if req.Count == 0 {
           ctx.JSON(400, gin.H{ "error": "La cantidad es requerida"})
           return
       }
       if req.Price == 0 {
           ctx.JSON(400, gin.H{ "error": "El precio es requerido"})
           return
       }
       p, err := c.service.Update(int(id), req.Name, req.Type, req.Count, req.Price)
       if err != nil {
           ctx.JSON(404, gin.H{ "error": err.Error() })
           return
       }
       ctx.JSON(200, p)
}}
```

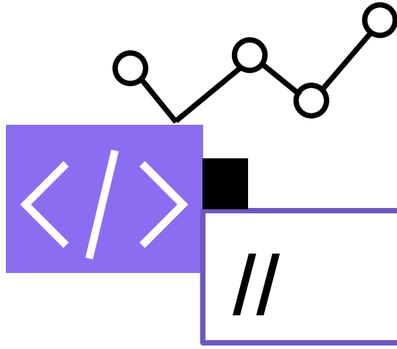


Paso 5 - Main del programa

Dentro del main del programa, se agrega el router correspondiente al método **PUT**.

```
{}
func main() {
    repo := products.NewRepository()
    service := products.NewService(repo)
    p := handler.NewProduct(service)
    r := gin.Default()
    pr := r.Group("/products")
    pr.POST("/", p.Store())
    pr.GET("/", p.GetAll())
    pr.PUT("/:id", p.Update())
    r.Run()
}
```

02 PATCH



¿Cómo podemos recibir una petición PATCH?

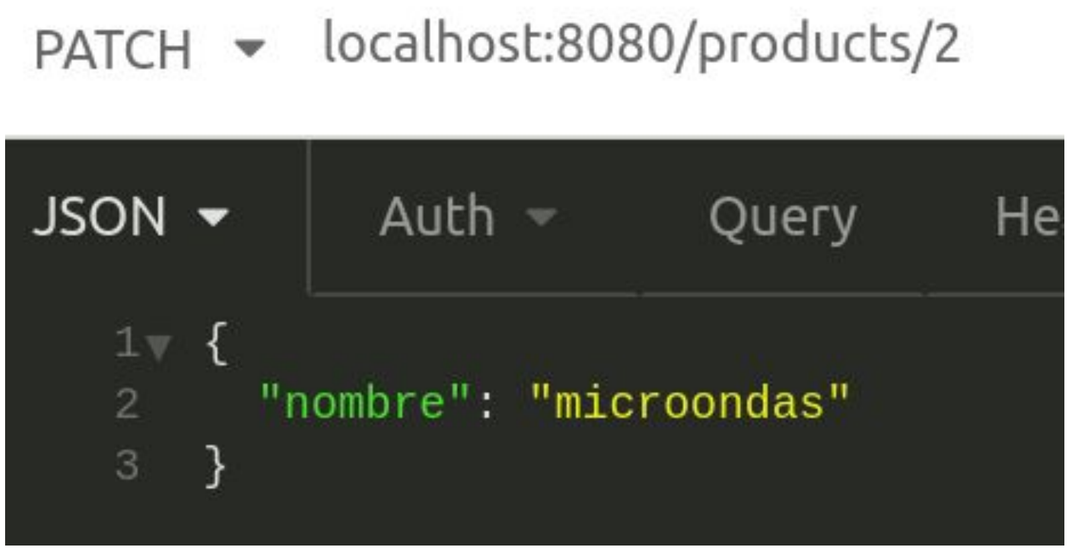
```
{}
```

```
pr.POST("/", p.Store())
pr.GET("/", p.GetAll())
pr.PUT("/:id", p.Update())
pr.PATCH("/:id", p.UpdateName())
```

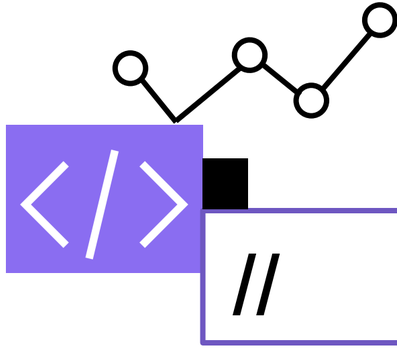
Petición/Respuesta

Se recibe una solicitud **PATCH** con un **path parameter** que indique el ID del producto almacenado que se está intentando modificar, y se modifica solo el campo nombre.

Petición:



Respuesta:

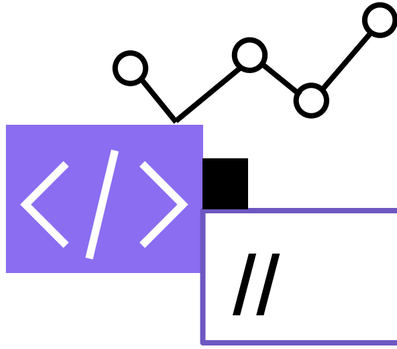


Paso 1 - Agregar método en interface

- Definir un servicio web mediante el método **PATCH**, el cual tendrá como path "**products/:id**".
- Agregar los métodos a utilizar en las interfaces de **Repository** y **Service**.

```
{}
type Repository interface {
   GetAll() ([]Product, error)
   Store(id int, name, productType string, count int, price float64) (Product,
error)
   LastID() (int, error)
   UpdateName(id int, name string) (Product, error)
   Update(id int,  name, productType string, count int, price float64) (Product,
error)
}
```

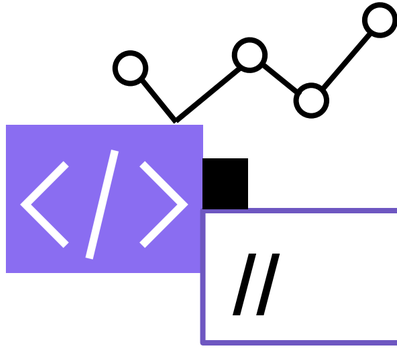
```
{}
type Service interface {
   GetAll() ([]Product, error)
   Store(name, productType string, count int, price float64) (Product, error)
   Update(id int, name, productType string, count int, price float64) (Product,
error)
   UpdateName(id int, name string) (Product, error)
}
```



Paso 2 - Repositorio

Se implementa la funcionalidad para actualizar el nombre del producto en memoria en caso de que coincida con el ID enviado. Caso contrario, retorna un error.

```
{}
func (r *repository) UpdateName(id int, name string) (Product, error) {
    var p Product
    updated := false
    for i := range ps {
        if ps[i].ID == id {
            ps[i].Name = name
            updated = true
            p = ps[i]
        }
    }
    if !updated {
        return Product{}, fmt.Errorf("Producto %d no encontrado", id)
    }
    return p, nil
}
```



Paso 3 - Servicio

Dentro del servicio se llama al repositorio para que proceda a actualizar el nombre del producto.

```
{}
```

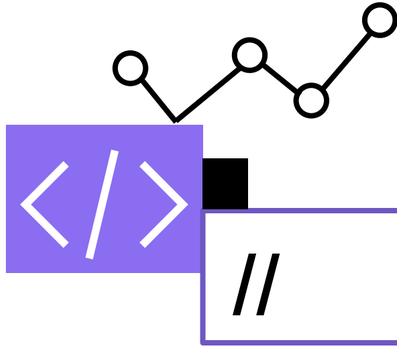
```go
func (s *service) UpdateName(id int, name string) (Product, error) {

   return s.repository.UpdateName(id, name)
}
```

Paso 4

Se agrega el controlador **UpdateName** en el **Handler** de productos.

```
func (c *Product) UpdateName() gin.HandlerFunc {
    return func(ctx *gin.Context) {
        token := ctx.GetHeader("token")
        if token != "123456" {
            ctx.JSON(401, gin.H{ "error": "token inválido" })
            return
        }
        id, err := strconv.ParseInt(ctx.Param("id"),10, 64)
        if err != nil {
            ctx.JSON(400, gin.H{ "error":  "invalid ID"})
            return
        }
        var req request
        if err := ctx.ShouldBindJSON(&req); err != nil {
            ctx.JSON(400, gin.H{ "error": err.Error() })
            return
        }
        if req.Name == "" {
            ctx.JSON(400, gin.H{ "error": "El nombre del producto es requerido"})
            return
        }
        p, err := c.service.UpdateName(int(id), req.Name)
        if err != nil {
            ctx.JSON(404, gin.H{ "error": err.Error() })
            return
        }
        ctx.JSON(200, p)
}}
```

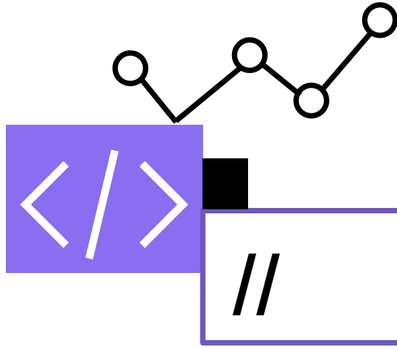


Paso 5 - Main del programa

Dentro del main del programa, se agrega el router correspondiente al método **PATCH**.

```
{}
func main() {
    repo := products.NewRepository()
    service := products.NewService(repo)
    p := handler.NewProduct(service)
    r := gin.Default()
    pr := r.Group("/products")
    pr.POST("/", p.Store())
    pr.GET("/", p.GetAll())
    pr.PUT("/:id", p.Update())
    pr.PATCH("/:id", p.UpdateName())
    r.Run()
}
```

03 DELETE



¿Cómo podemos recibir una petición DELETE?

```
{}
```

```
pr.POST("/", p.Store())
pr.GET("/", p.GetAll())
pr.PUT("/:id", p.Update())
pr.PATCH("/:id", p.UpdateName())
pr.DELETE("/:id", p.Delete())
```

Petición/Respuesta

Se recibe una solicitud **DELETE** con un **path parameter** que indica el ID del producto almacenado que se está intentando eliminar.

Petición:

- No necesita tener datos en su **body**.
- Solo precisa indicar el ID del producto que se desea eliminar para identificarlo correctamente.
- Se pasará como parámetro de ruta.

Respuesta:

- Si al procesar la petición, el servidor encuentra el ID especificado, elimina el producto y devuelve un status 2xx.

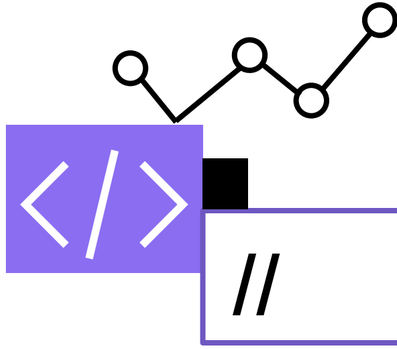
La respuesta, en su **body**, puede contener un detalle actualizado de los productos para constatar que el indicado fue eliminado.

Paso 1 - Agregar método en interface

- Se define un servicio web mediante el método **DELETE**, el cual tiene como path "**products/:id**".
- Agregar los métodos a utilizar en las interfaces de **Repository** y **Service**.

```
{}
type Repository interface {
   GetAll() ([]Product, error)
   Store(id int, name, productType string, count int, price float64) (Product, error)
   LastID() (int, error)
   UpdateName(id int, name string) (Product, error)
   Update(id int,  name, productType string, count int, price float64) (Product, error)
   Delete(id int) error
}
```

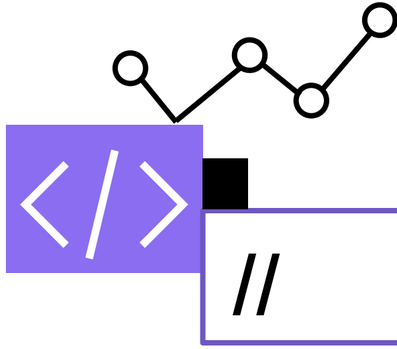
```
{}
type Service interface {
   GetAll() ([]Product, error)
   Store(name, productType string, count int, price float64) (Product, error)
   Update(id int, name, productType string, count int, price float64) (Product, error)
   UpdateName(id int, name string) (Product, error)
   Delete(id int) error
}
```



Paso 2 - Repositorio

Se implementa la funcionalidad para eliminar el producto en memoria en caso de que coincida con el ID enviado. Caso contrario, retorna un error.

```
{}
func (r *repository) Delete(id int) error {
    deleted := false
    var index int
    for i := range ps {
        if ps[i].ID == id {
            index = i
            deleted = true
        }
    }
    if !deleted {
        return fmt.Errorf("Producto %d no encontrado", id)
    }
    ps = append(ps[:index], ps[index+1:]...)
    return nil
}
```



Paso 3 - Servicio

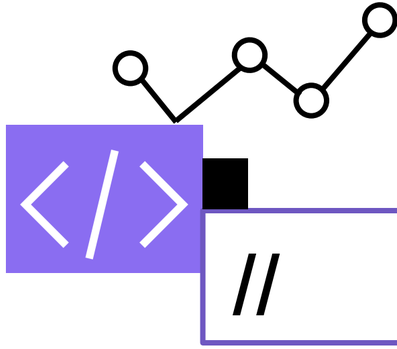
Dentro del servicio se llama al repositorio para que proceda a eliminar el producto.

```
{}
func (s *service) Delete(id int) error {
    return s.repository.Delete(id)
}
```

Paso 4

Se agrega el controlador **Delete** en el **Handler** de productos.

```
func (c *Product) Delete() gin.HandlerFunc {
    return func(ctx *gin.Context) {
        token := ctx.GetHeader("token")
        if token != "123456" {
            ctx.JSON(401, gin.H{ "error": "token inválido" })
            return
        }
        id, err := strconv.ParseInt(ctx.Param("id"),10, 64)
        if err != nil {
            ctx.JSON(400, gin.H{ "error":  "invalid ID"})
            return
        }
        err = c.service.Delete(int(id))
        if err != nil {
            ctx.JSON(404, gin.H{ "error": err.Error() })
            return
        }
        ctx.JSON(200, gin.H{ "data": fmt.Sprintf("El producto %d ha sido
eliminado", id) })
    }
}
```



Paso 5 - Main del programa

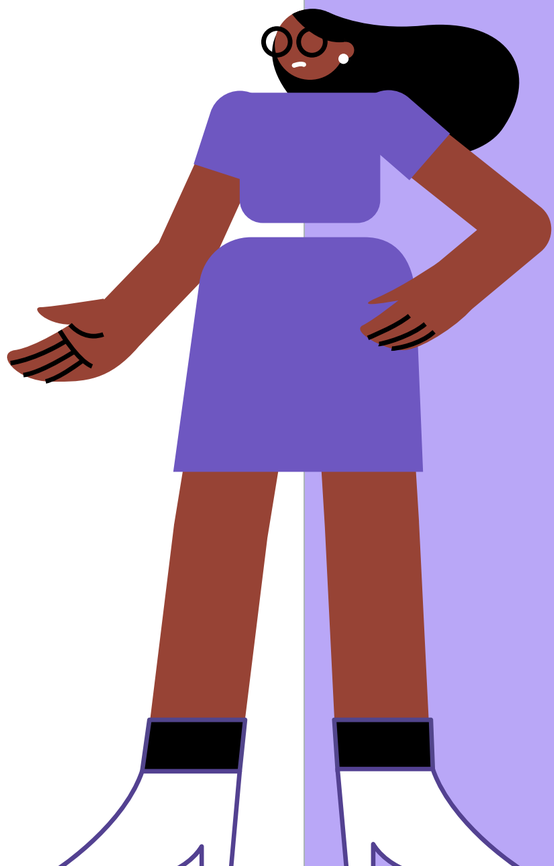
Dentro del main del programa, se agrega el router correspondiente al método **DELETE**.

```
{}
func main() {
    repo := products.NewRepository()
    service := products.NewService(repo)
    p := handler.NewProduct(service)
    r := gin.Default()
    pr := r.Group("/products")
    pr.POST("/", p.Store())
    pr.GET("/", p.GetAll())
    pr.PUT("/:id", p.Update())
    pr.PATCH("/:id", p.UpdateName())
    pr.DELETE("/:id", p.Delete())
    r.Run()
}
```

Conclusiones

En esta clase aprendimos los conceptos teóricos de las peticiones HTTP: **PUT**, **PATCH** y **DELETE**.

Además, aprendimos cómo aplicarlas en Go para hacer que nuestras aplicaciones puedan interactuar correctamente con sus usuarios.



DIGITALHOUSE

¡Muchas gracias!