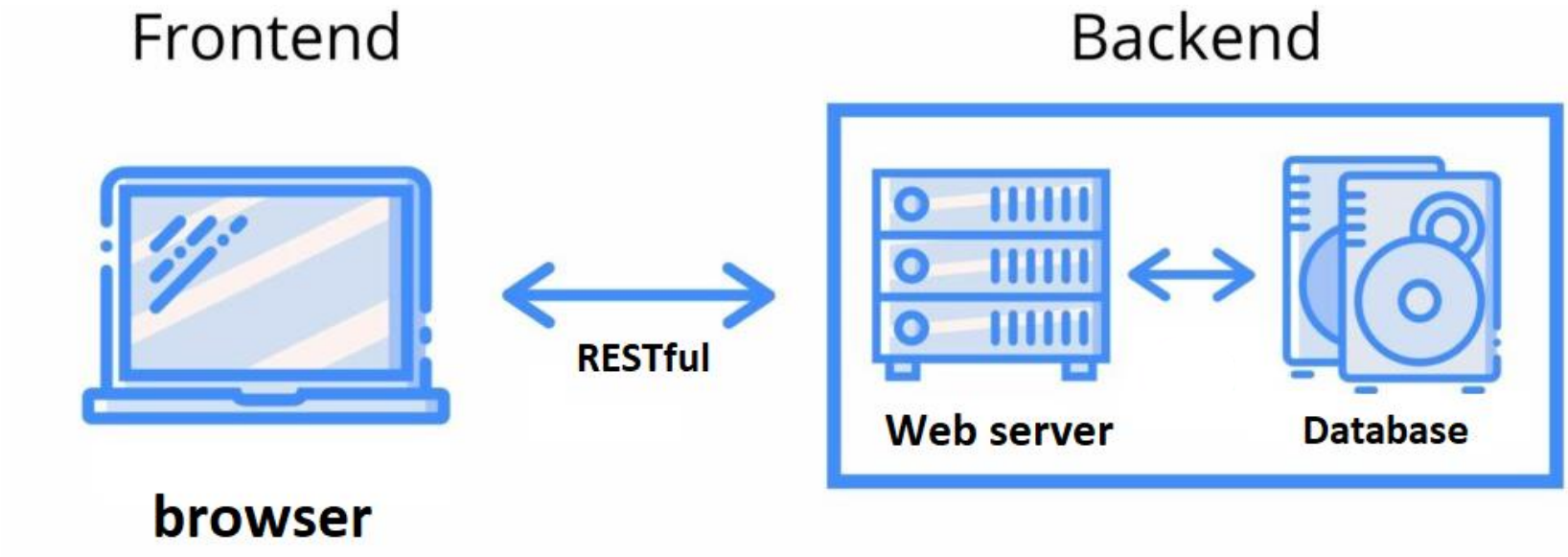
Primeros pasos en la Autenticación

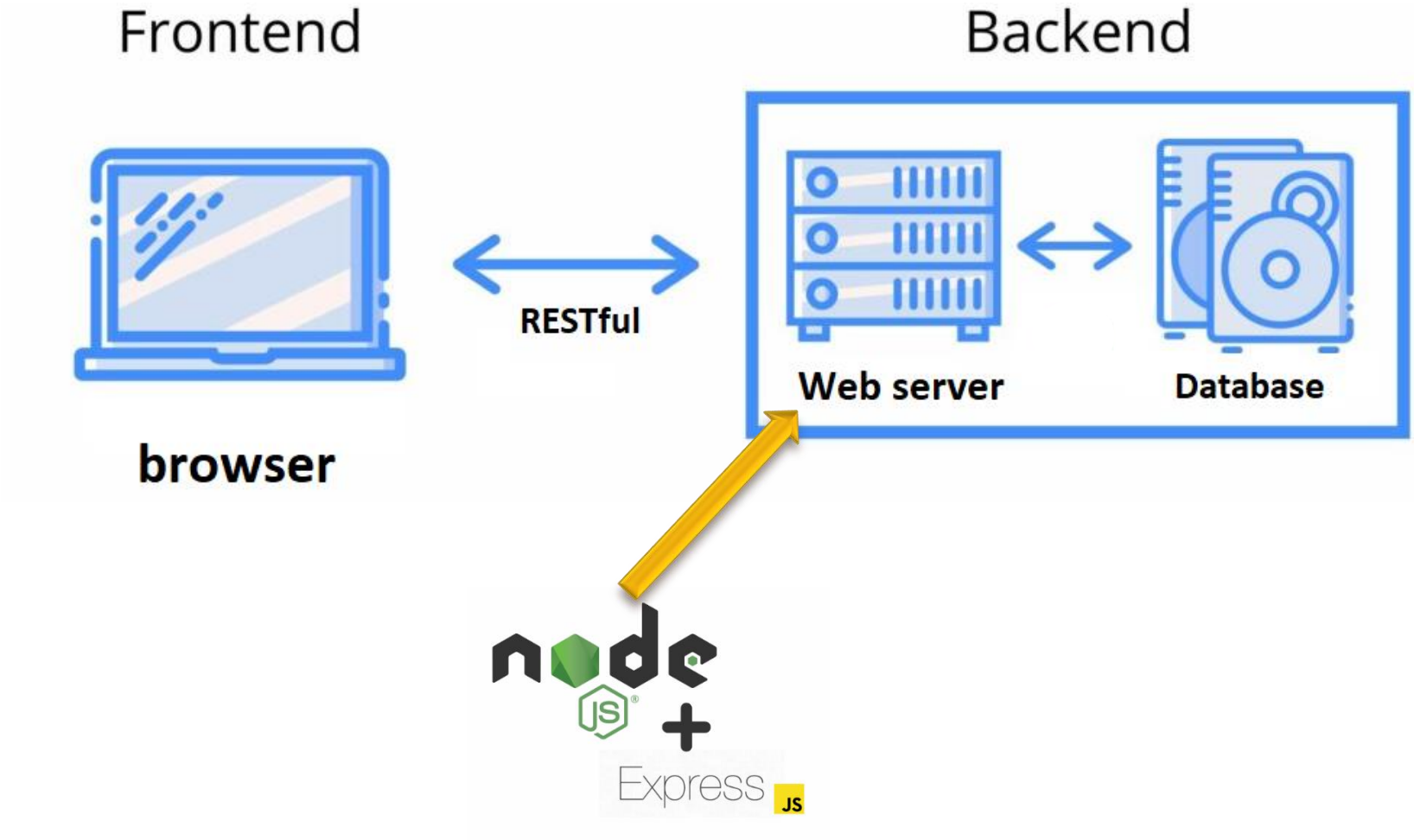
Autenticación básica, Tokens, Token Bearer, JWT, OAuth2, https…

Antes de nada veamos de donde venimos y a donde vamos

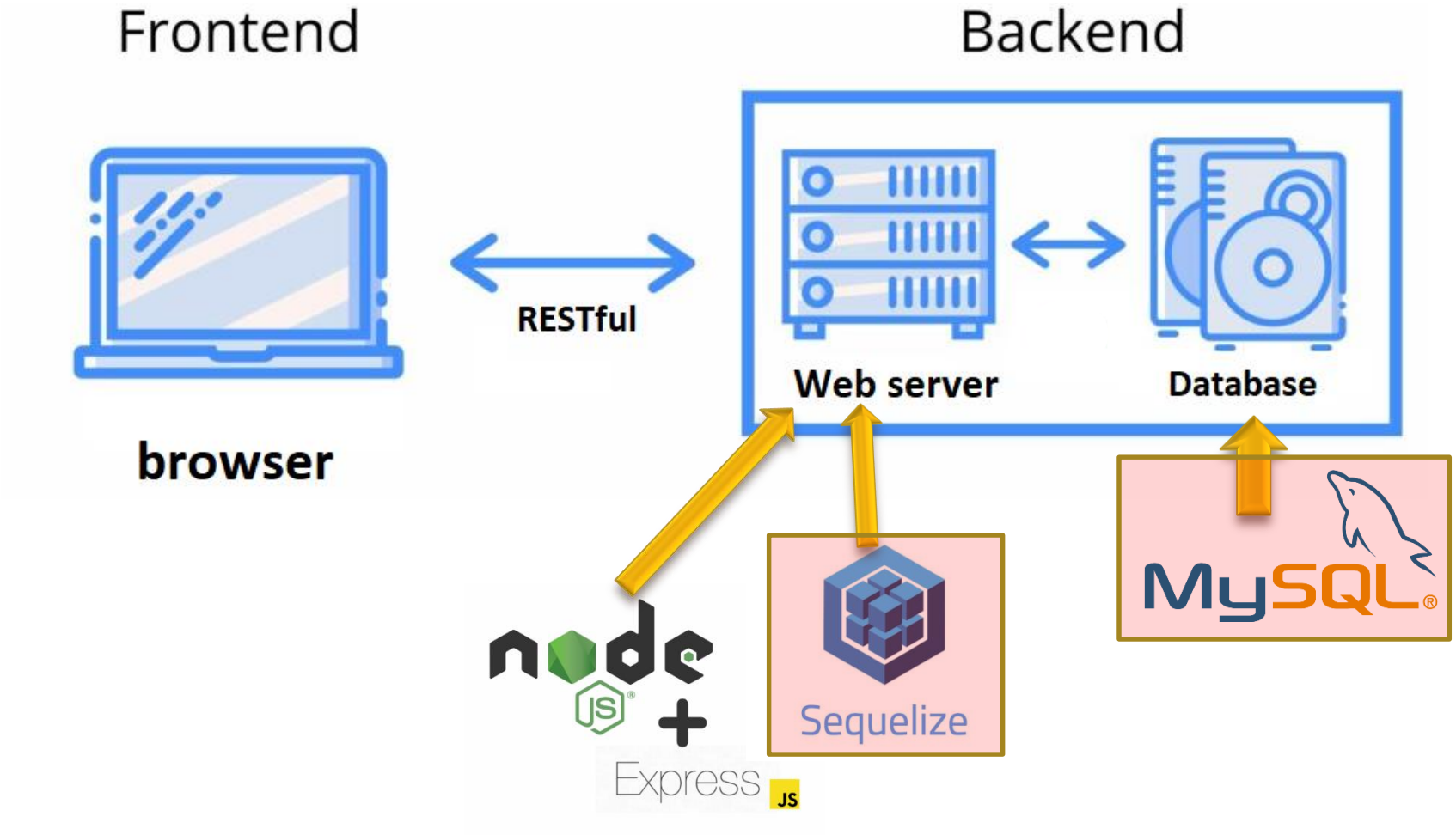
¿Qué hemos hecho hasta ahora?



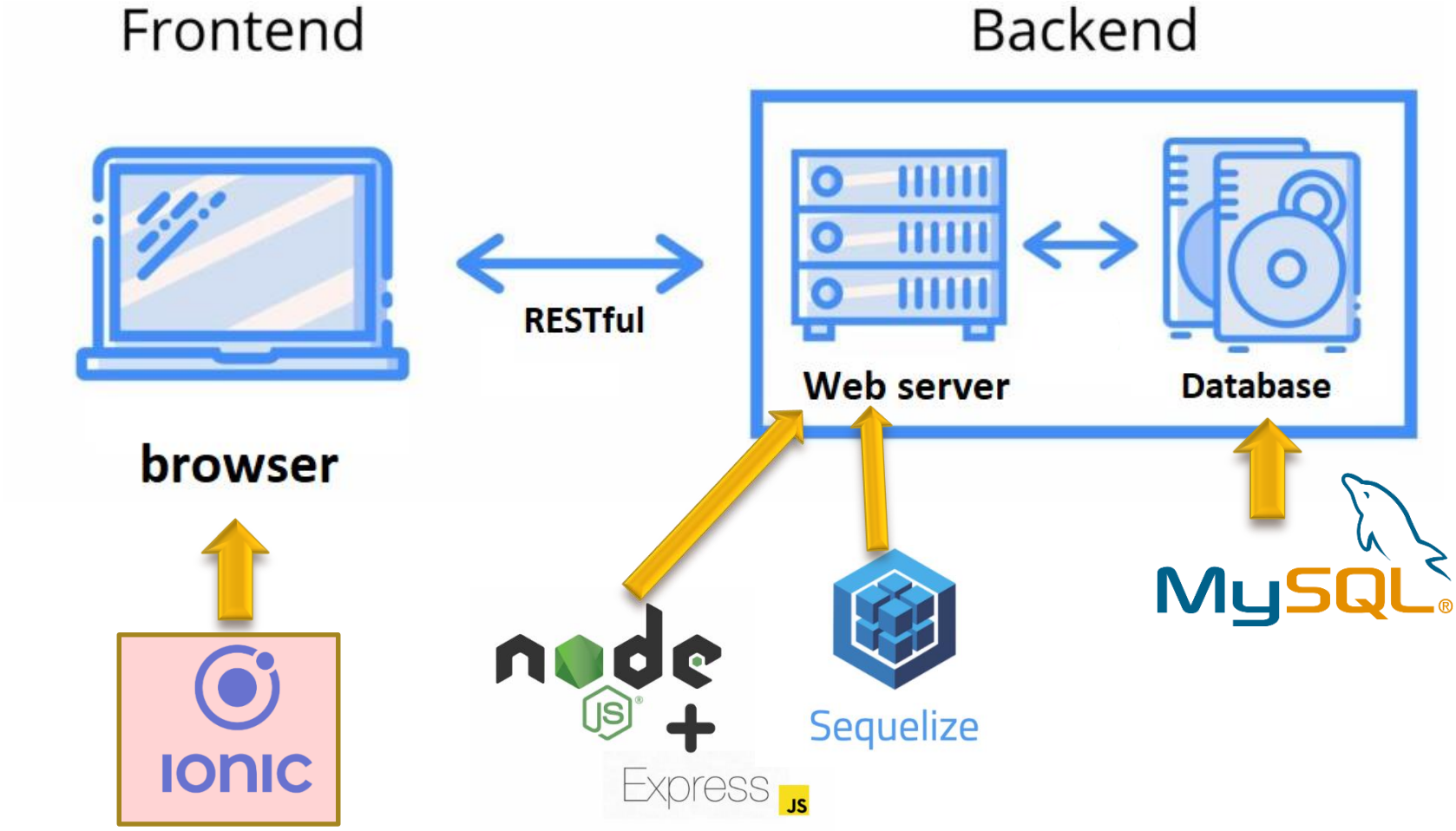
¿Qué hemos hecho hasta ahora?
(end-points en el backend)



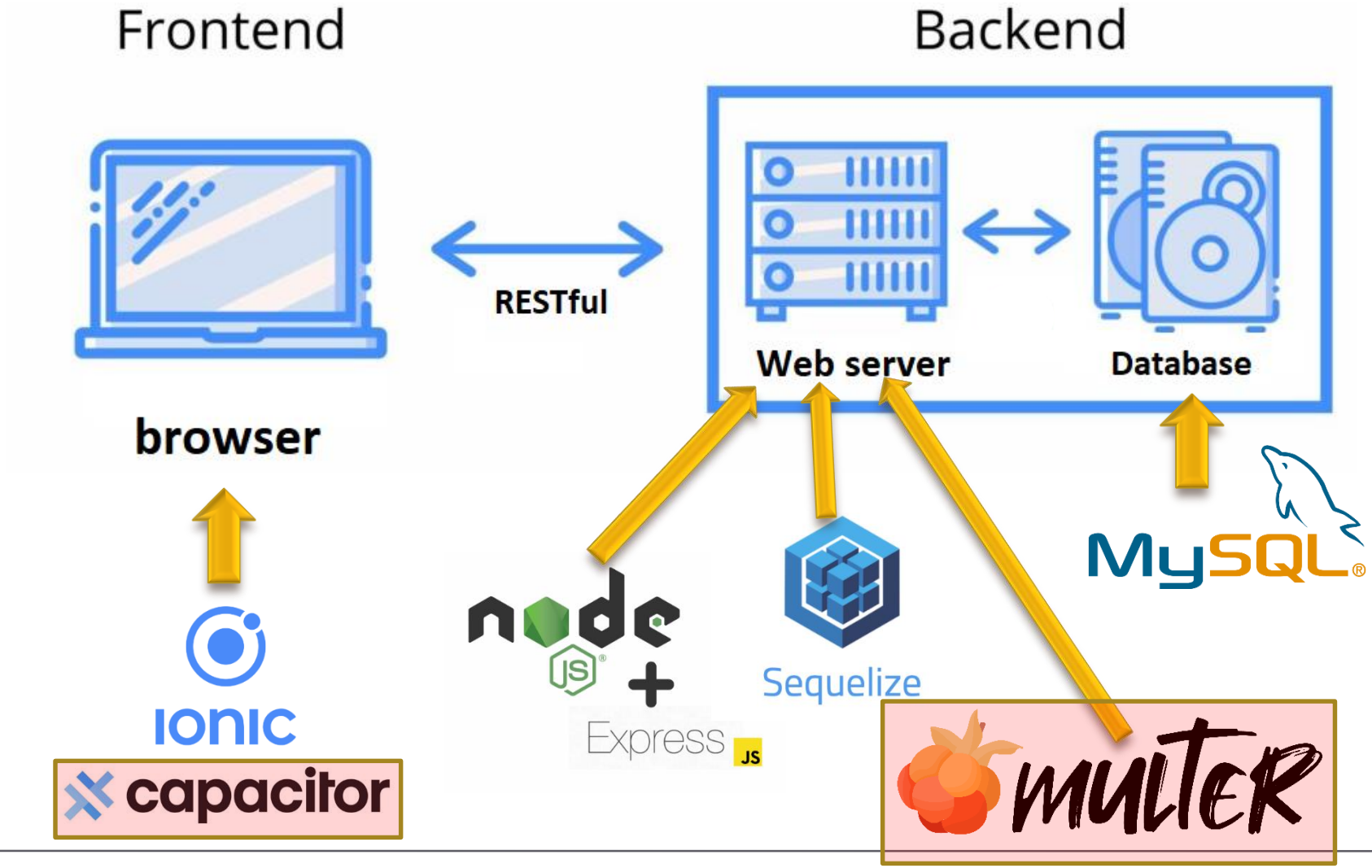
¿Qué hemos hecho hasta ahora? (Accedemos a la BD usando un ORM)



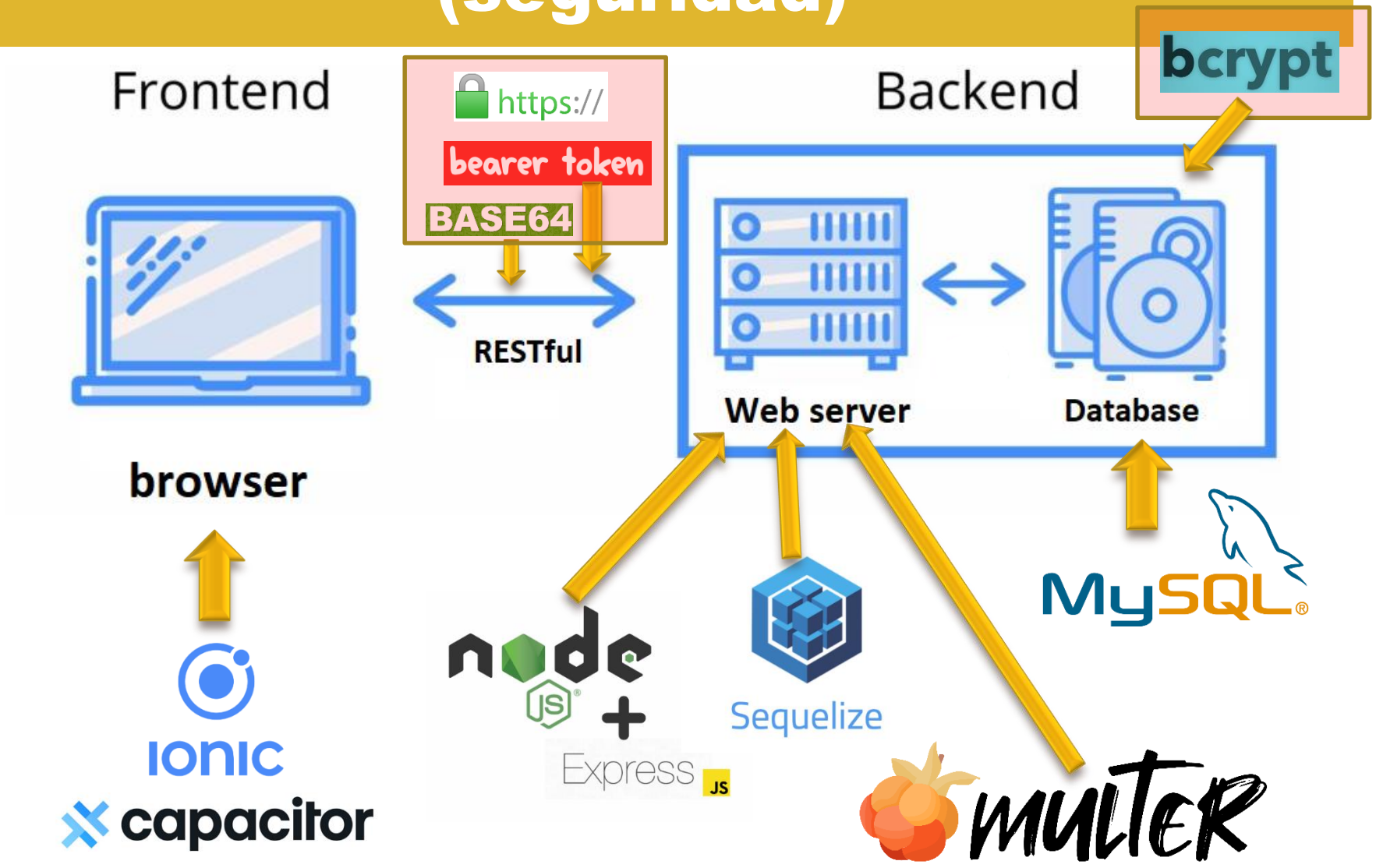
¿Qué hemos hecho hasta ahora?
(Creamos el frontend y consumimos la API)



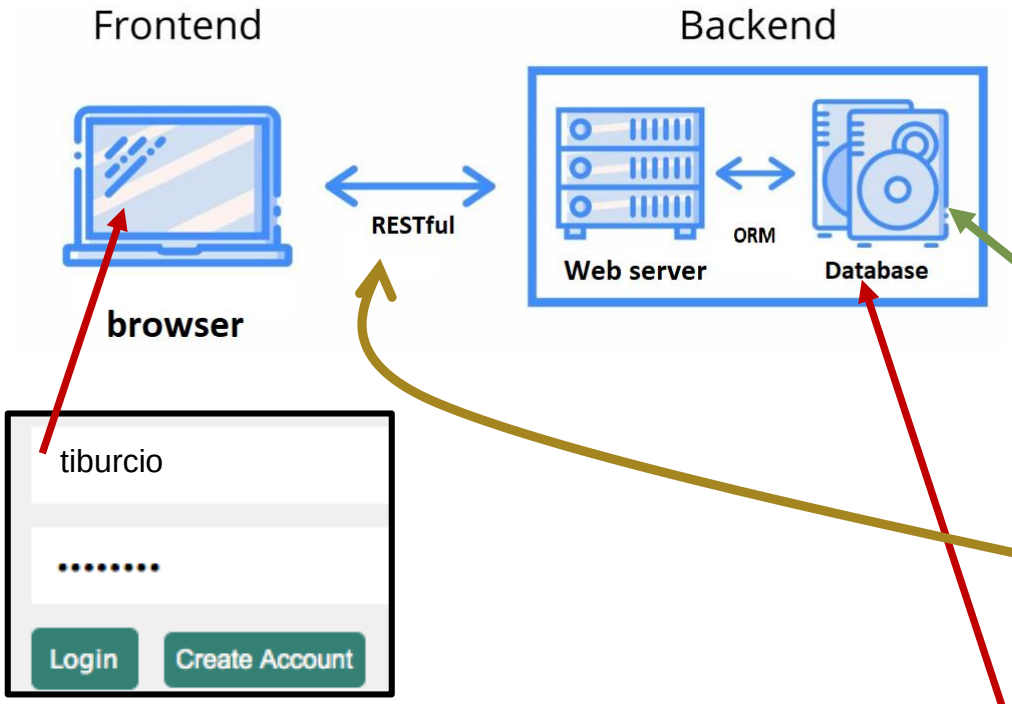
¿Qué hemos hecho hasta ahora? (Añadimos imágenes)



¿Qué vamos a hacer ahora? (seguridad)



¿Qué queremos conseguir?



En resumen que el usuario pueda hacer un login y un sign up y que sea lo más seguro posible

Por un lado que la contraseña se almacene de forma encriptada

Por otro lado que la información viaje segura

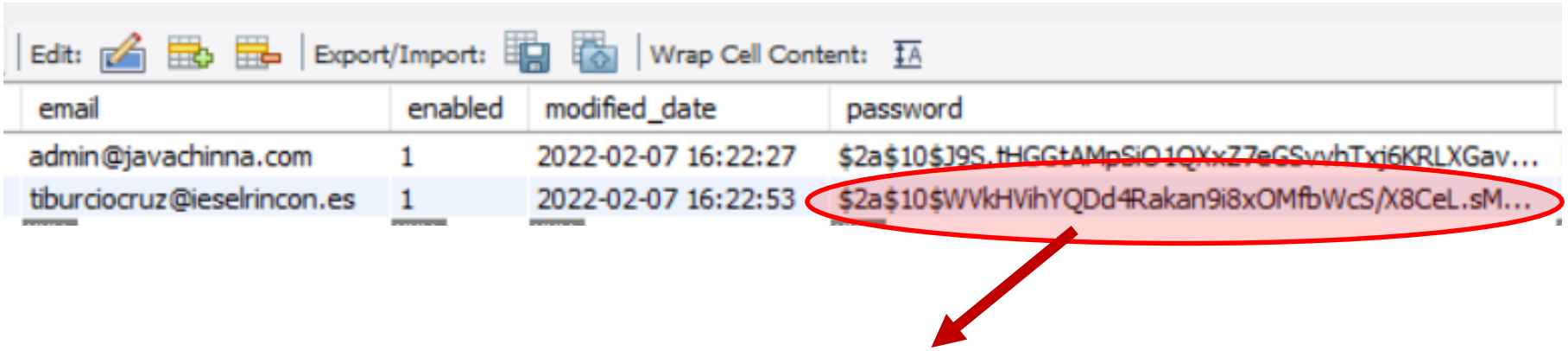
Edit: | Export/Import: | Wrap Cell Content:

email	enabled	modified_date	password
admin@javachinna.com	1	2022-02-07 16:22:27	$2a$10$19S.tHGGtAMpSiO1QXxZ7eGSvvhTxj6KRLXGav...
tiburciocruz@ieselrincon.es	1	2022-02-07 16:22:53	$2a$10$WVkHVihYQDd4Rakan9i8xOMfbWcS/X8CeL.sM...

Veamos Bcrypt en detalle

Bcrypt

Objetivo: Guardar las contraseñas encriptadas en la BD para que ni siquiera el Administrador de la BD conozca las contraseñas de los usuarios.

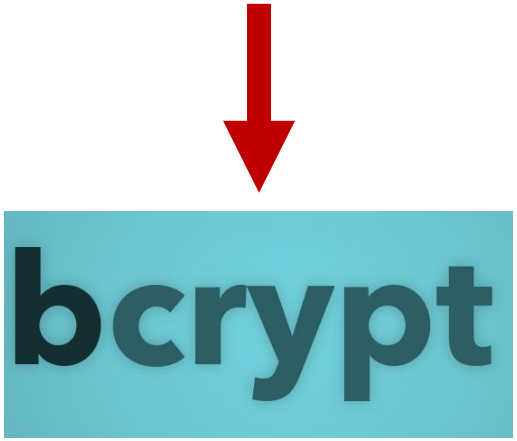


Aunque se comprometa la BD resulta imposible conocer la contraseña

Bcrypt
¿Cómo funciona?

Cuando el usuario se **registra** se encripta la contraseña y se introduce encriptada en la BD

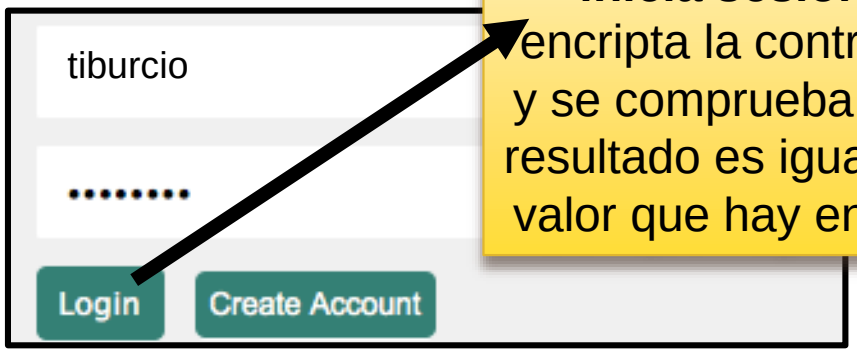
Mi clave molona



Edit: Export/Import: Wrap Cell Content:

email	enabled	modified_date	password
admin@javachinna.com	1	2022-02-07 16:22:27	$2a$10$J9S.tHGGtAMpSiO1QXxZ7eGSvvhTxi6KRLXGav...
tiburciocruz@ieselrincon.es	1	2022-02-07 16:22:53	$2a$10$WVkHVihYQDd4Rakan9i8xOMfbWcS/X8CeL.sM...

Bcrypt
¿Cómo funciona?



Cuando el usuario **inicia sesión** se encripta la contraseña y se comprueba que el resultado es igual al del valor que hay en la BD

Mi clave molona

bcrypt

$2a$10$WVkHVihYQDd4Rakan9i8xOMfbWcS/X8CeL.sM...

¿Son iguales?

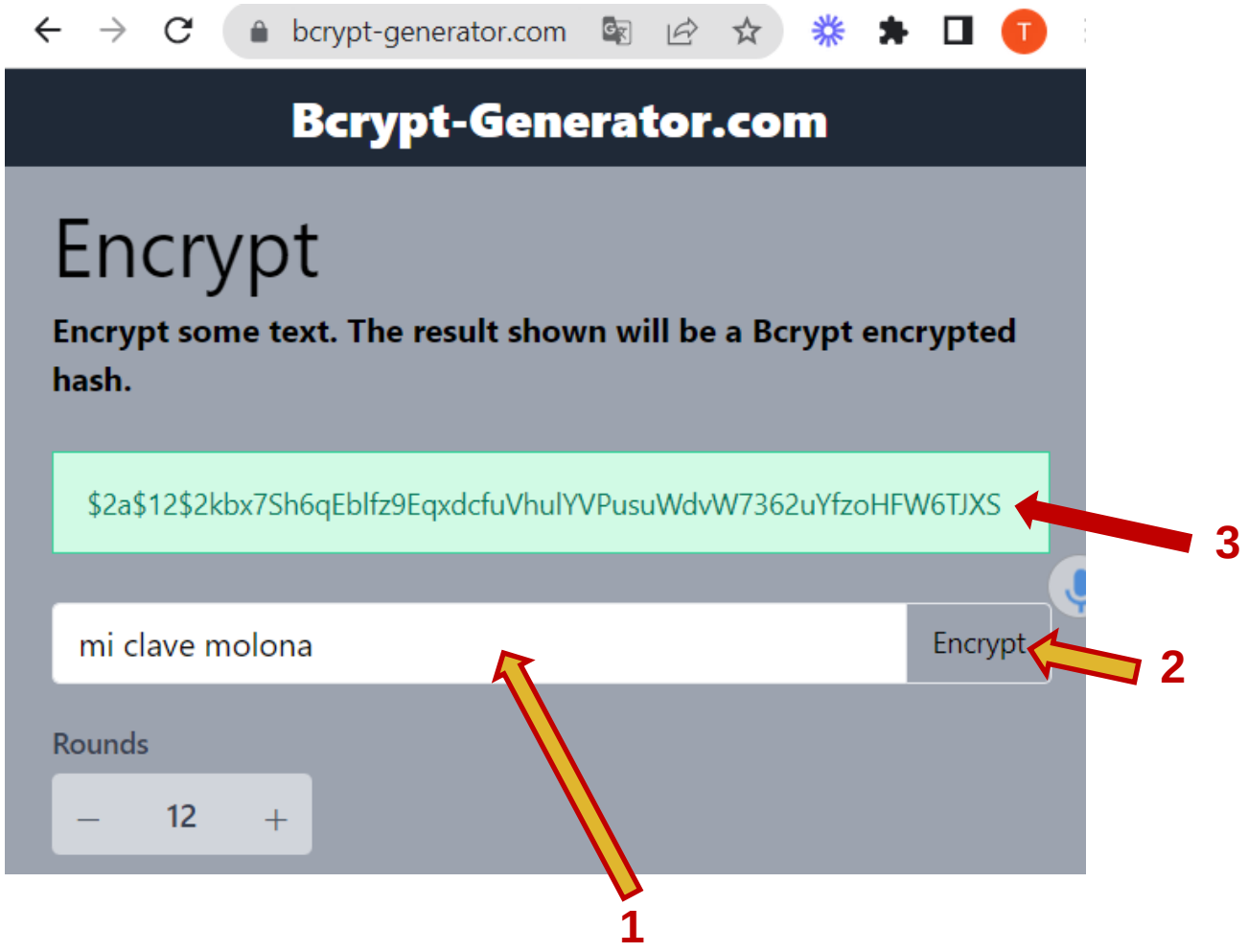
$2a$10$WVkHVihYQDd4Rakan9i8xOMfbWcS/X8CeL.sM...

Así la contraseña en claro no se guarda en la BD

Edit: Export/Import: Wrap Cell Content:

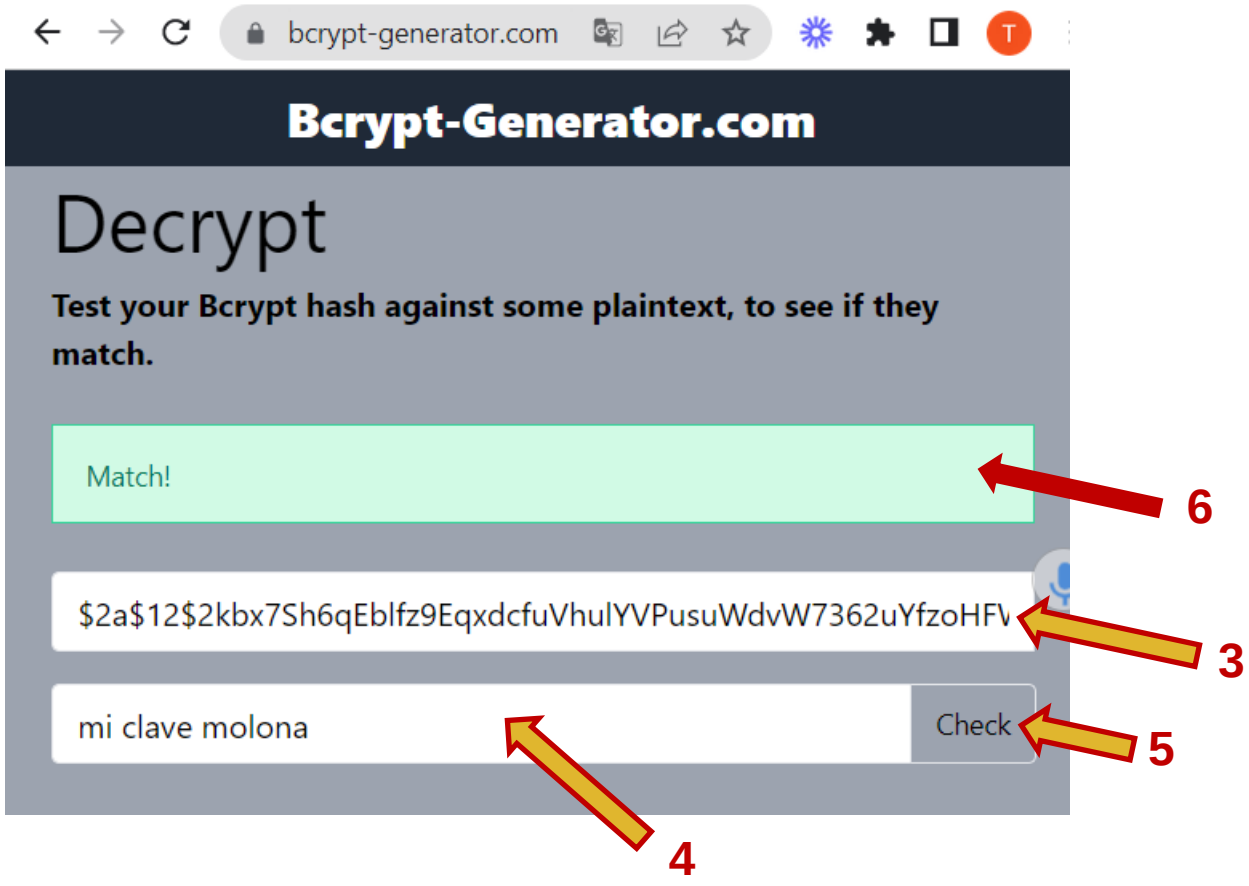
email	enabled	modified_date	password
admin@javachinna.com	1	2022-02-07 16:22:27	$2a$10$J9S.tHGGtAMpSiO1QXkZ7eGSvvhTxj6KRLXGav...
tiburciocruz@ieselrincon.es	1	2022-02-07 16:22:53	$2a$10$WVkHVihYQDd4Rakan9i8xOMfbWcS/X8CeL.sM...

Veámoslo en la práctica (https://bcrypt-generator.com/)



Introducimos la clave a encriptar (1), y al hacer clic en el botón Encrypt (2) creamos el hash (3)

Veámoslo en la práctica (https://bcrypt-generator.com/)

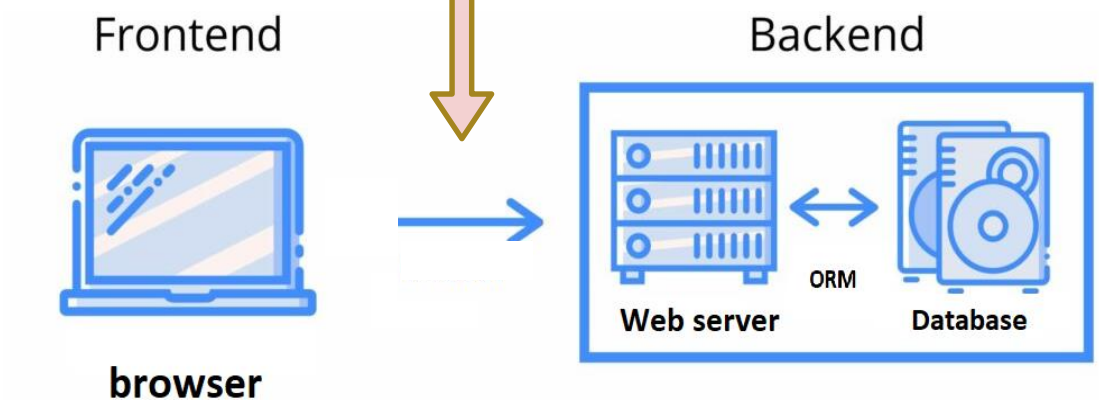


Introducimos el hash guardado en la BD (3), y la clave que suministra el usuario en el login (4). Al hacer clic en el botón Check (5) se comprueba si coinciden (6)

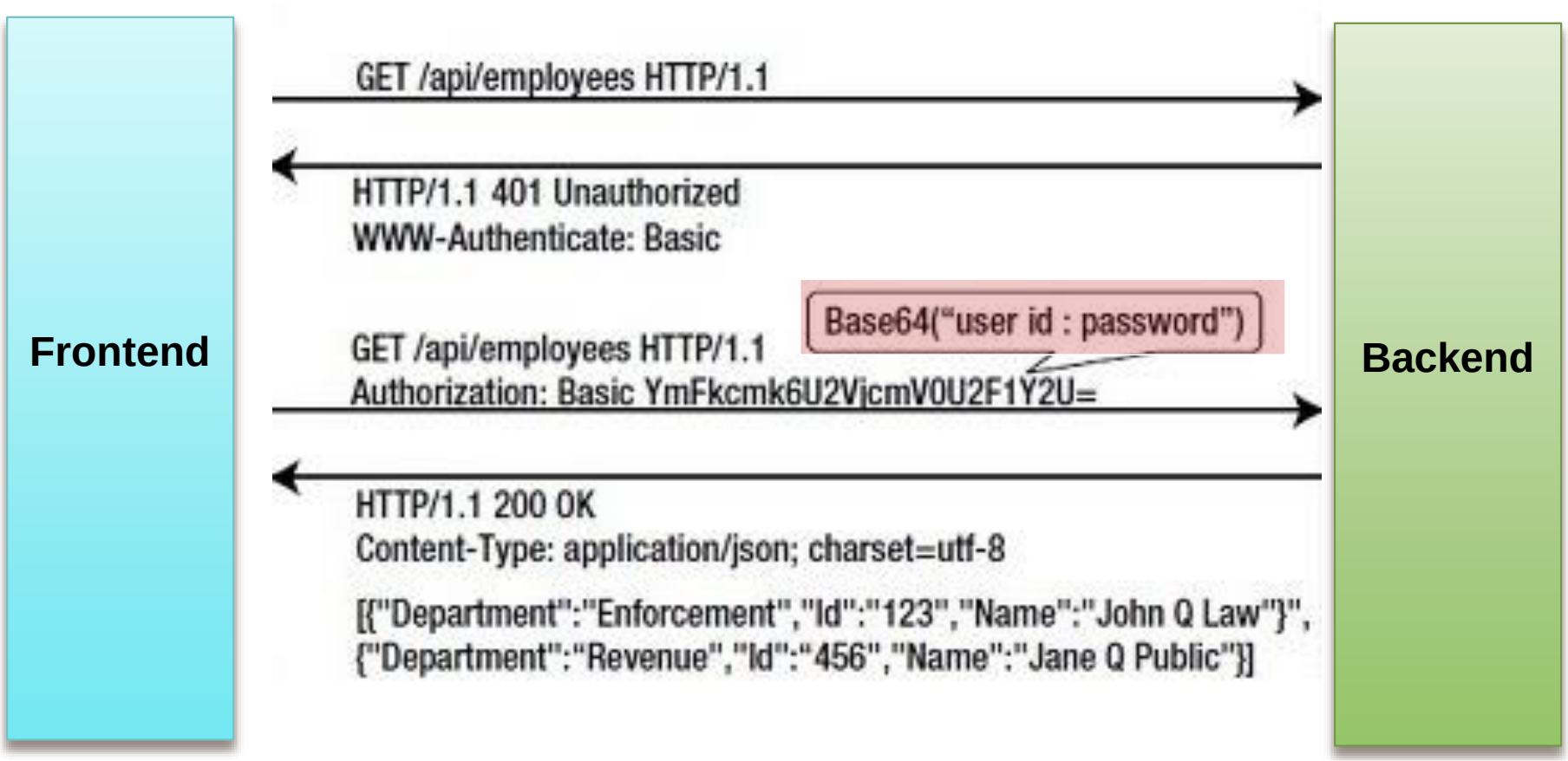
Veamos
Base64
en detalle

Base64
¿Cómo funciona?

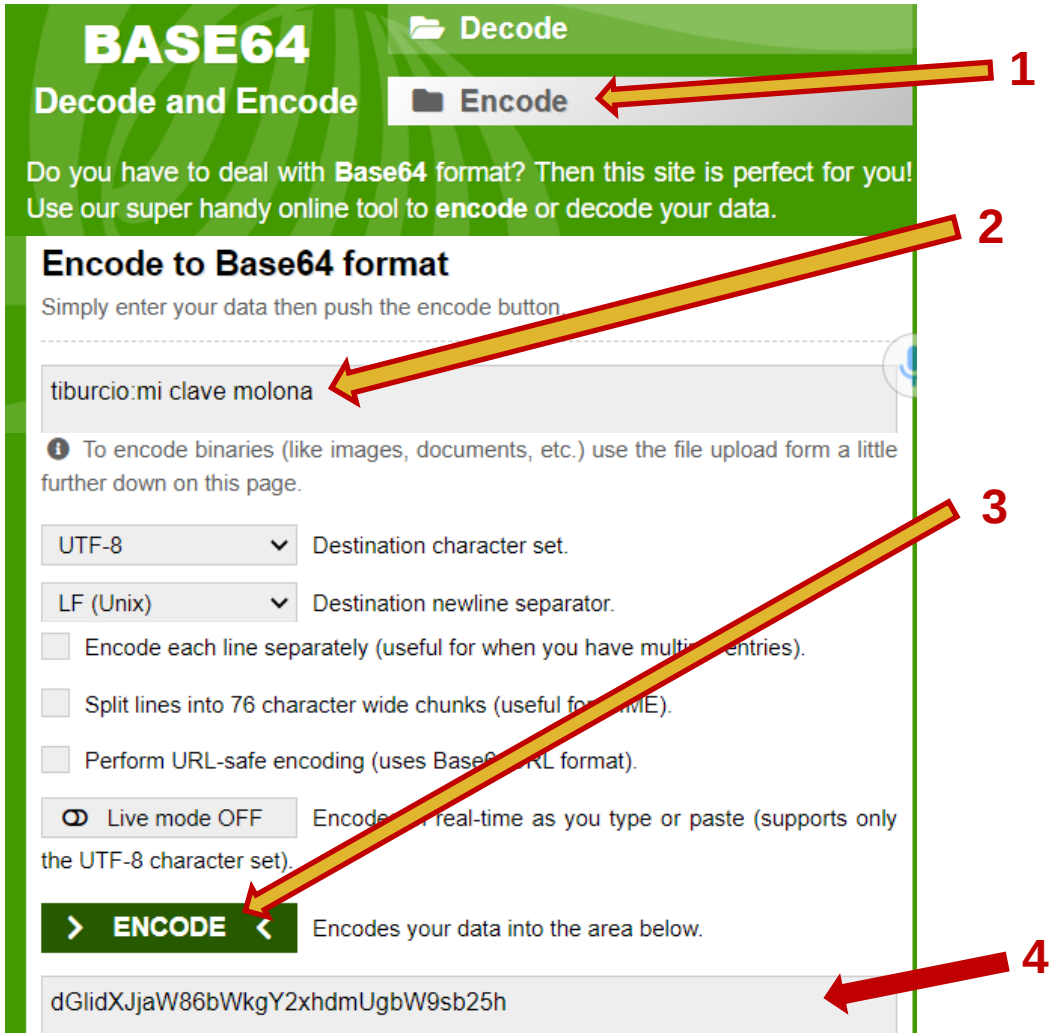
En el registro y en el inicio de sesión el **usuario:contraseña** viajan codificados en Base64



Base64
¿Cómo funciona?



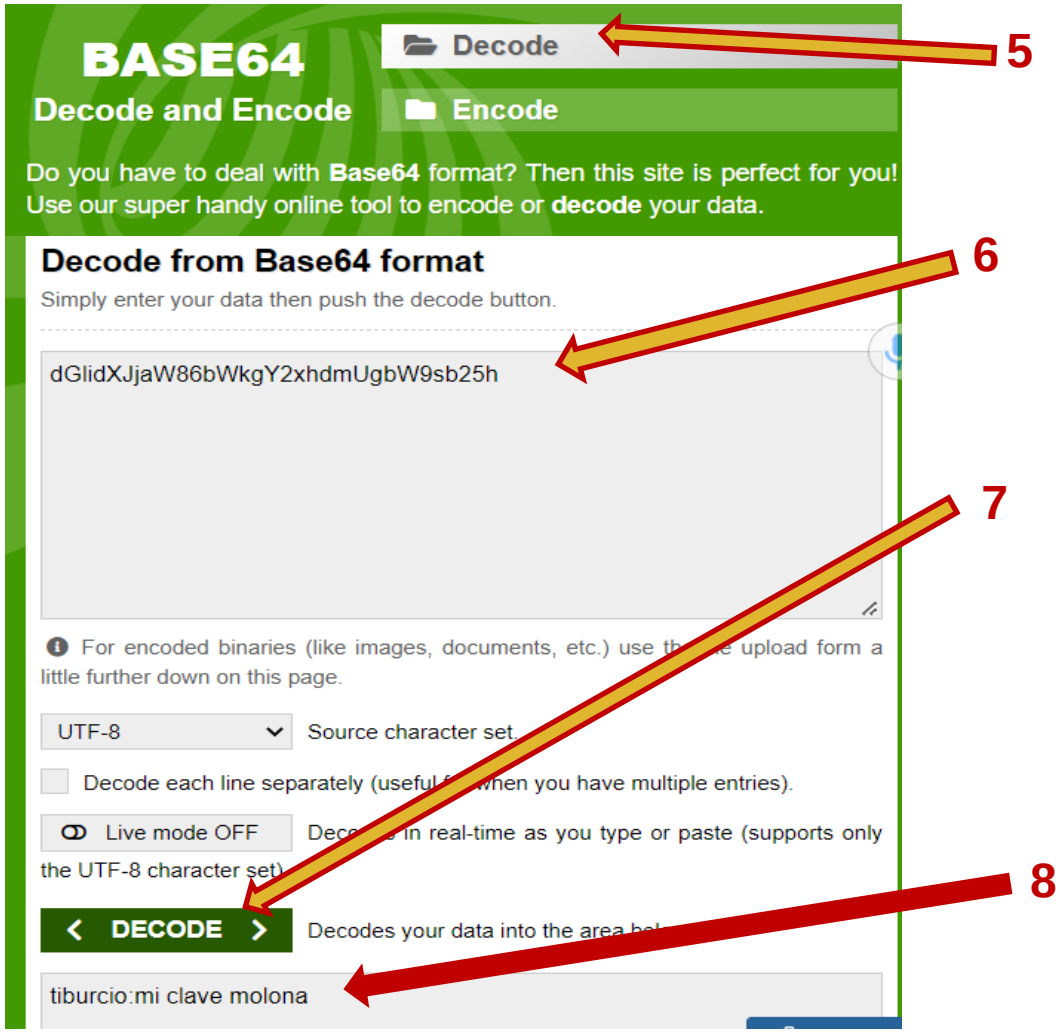
Veámoslo en la práctica (https://www.base64decode.org/)



Indicamos que queremos codificar (1), introducimos el usuario y la clave que suministra el usuario en el login separado por ":" (2). Al hacer clic en el botón Encode (3) se crea la codificación en base64 (4)

Observa que se introduce la credencial en la forma siguiente: ***usuario:contraseña***

Veámoslo en la práctica (https://www.base64decode.org/)



Indicamos que queremos descodificar (5), introducimos la información codificada (6). Al hacer clic en el botón DECODE (7) se obtiene la combinación usuario:contraseña (8)

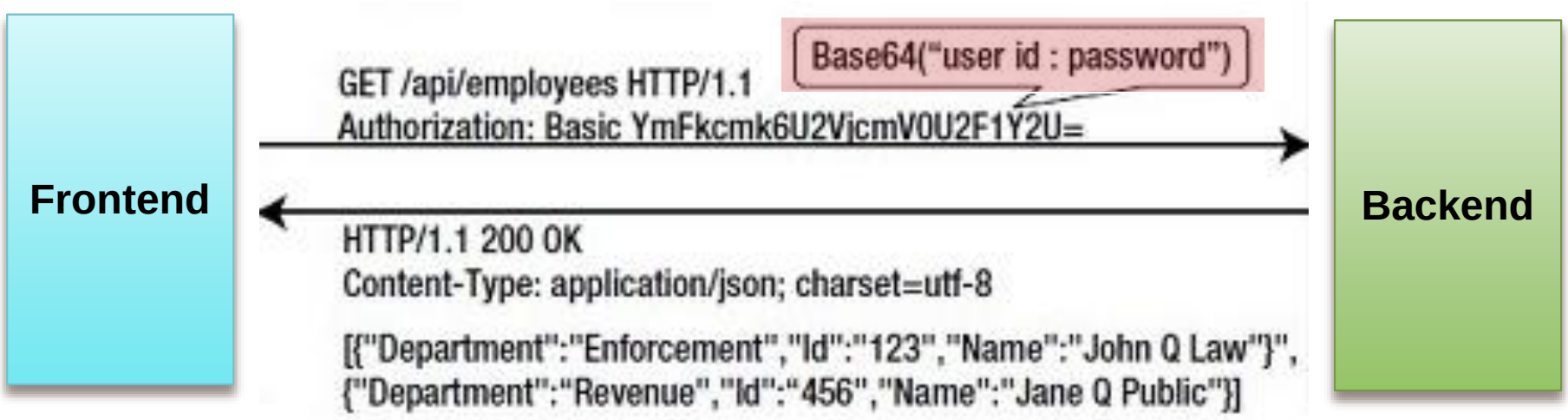
Observa que se obtienen las credenciales en la forma siguiente: ***usuario:contraseña***

Base64 es para codificar
y no para encriptar.

Enviar codificado en base64 no implica ninguna seguridad

Cualquiera puede descodificar las credenciales enviadas con base64

Base64
Autenticación Básica



A esto se le conoce como ***Autenticación Básica***

Solo se codifican la credenciales y no se consigue seguridad. Solo sirve para ocultar las credenciales cuando se hace el registro y el inicio de sesión. Y también cuando se incluyen para cada petición posterior.

¿Y entonces?

¿Cómo conseguimos la seguridad?

La seguridad en la autenticación básica la conseguimos con HTTPs

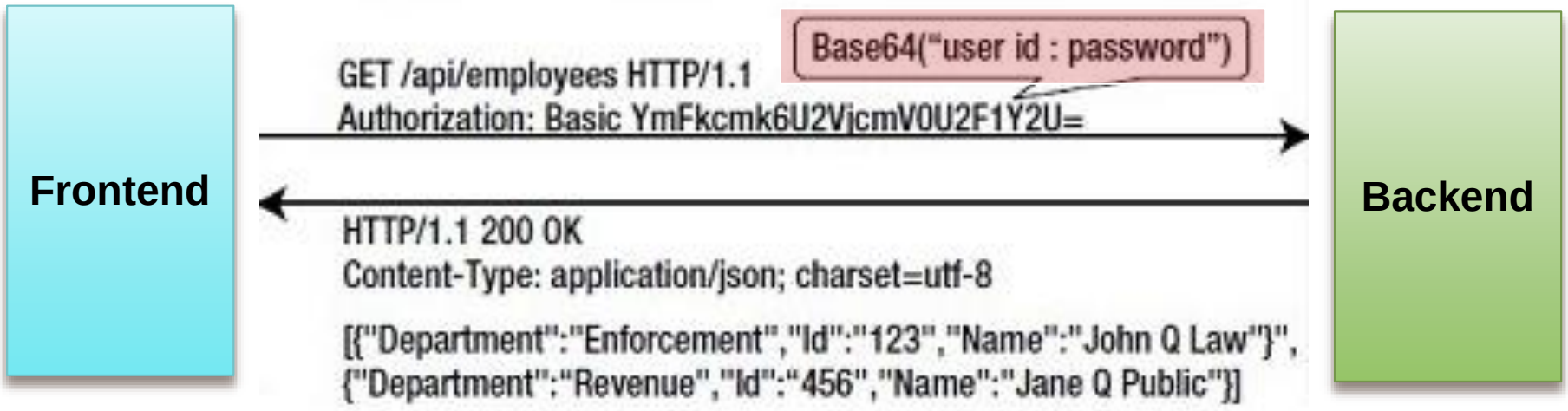
Si quieres aprender más sobre HTTPs puedes encontrar una práctica interesante en la plataforma

Pero aún hay más…

Veamos ahora el uso de tokens

Tokens

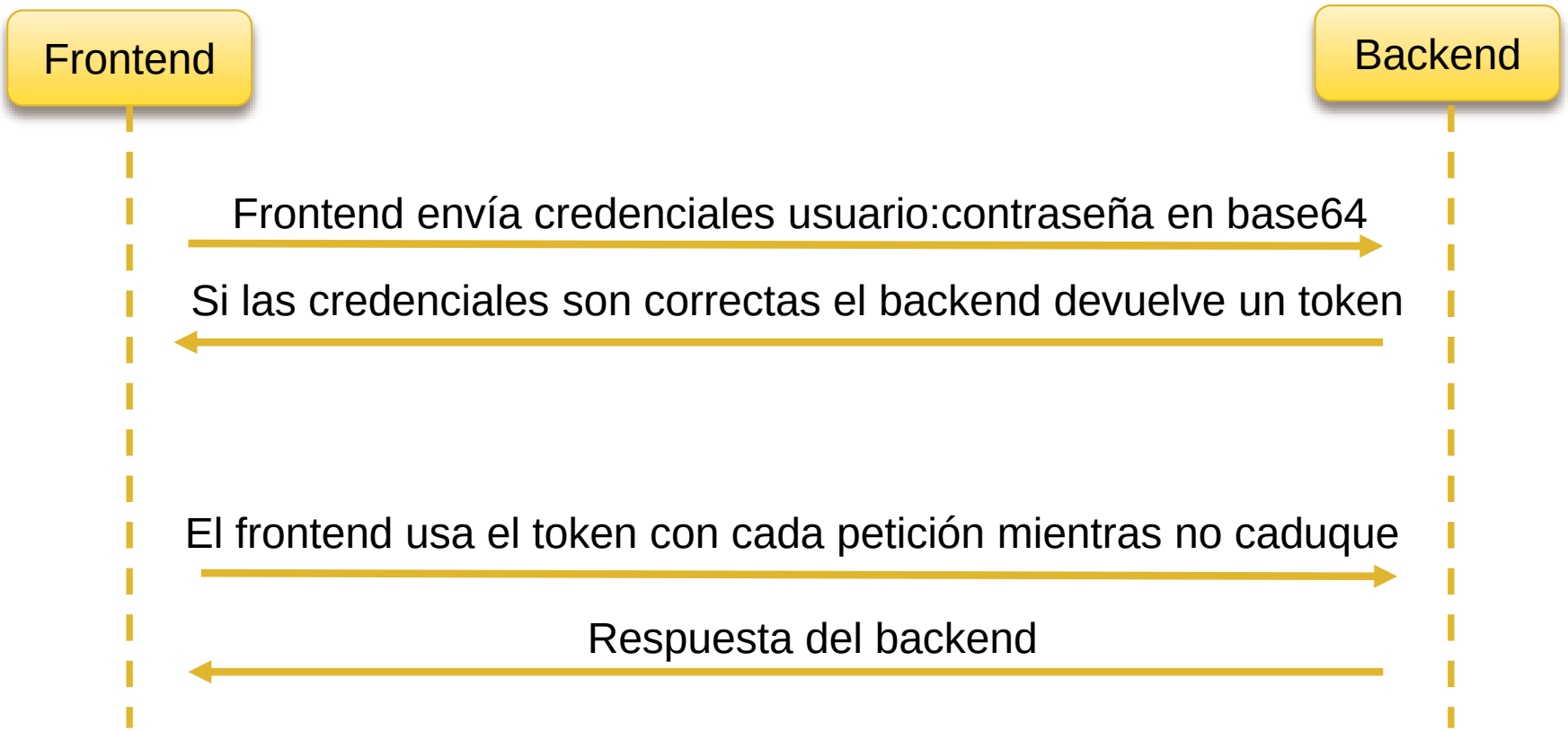
¿Qué mejora supone con respecto a la autenticación básica?



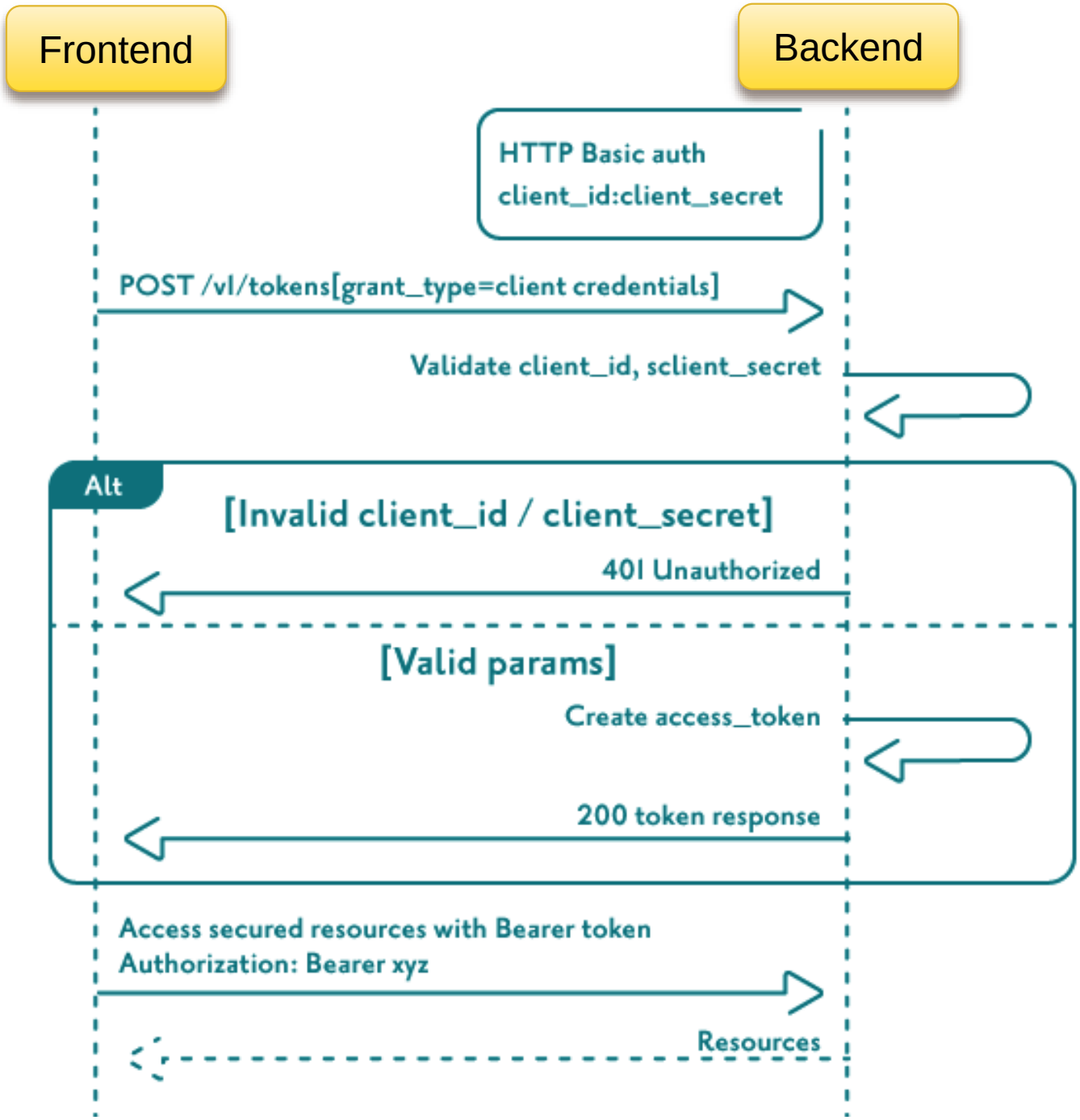
Ya hemos visto que esto es *Autenticación Básica.* En la autenticación básica cada vez que el frontend accede al backend envía sus credenciales codificadas en *Base64*.

No es adecuado enviar las credenciales todo el rato. Para solucionarlo cuando se hace el login el backend devuelve un token que es el que se utiliza para el resto de la sesión.

los tokens se suelen usar conjuntamente con la autenticación básica

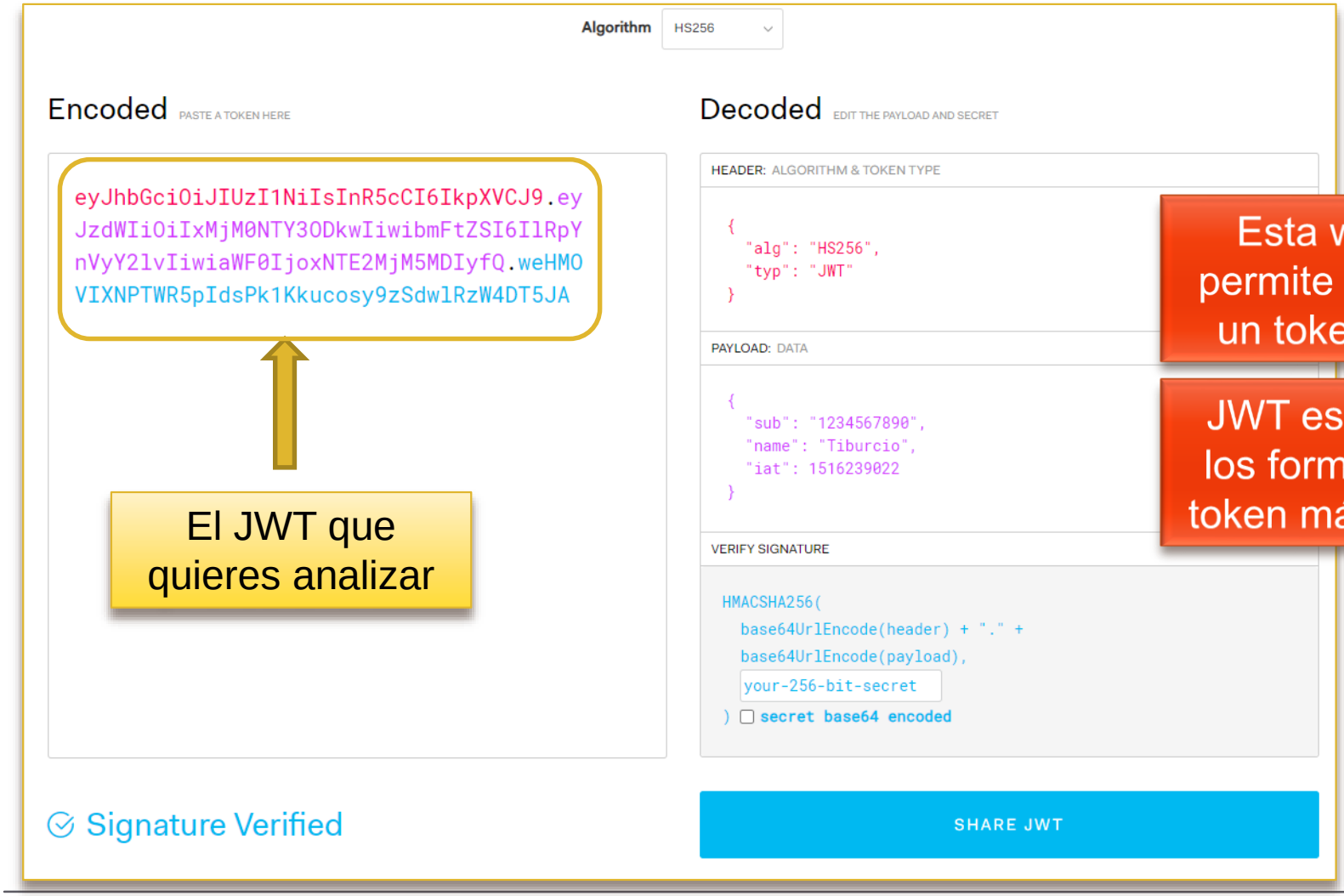


Veámoslo con más detalle…

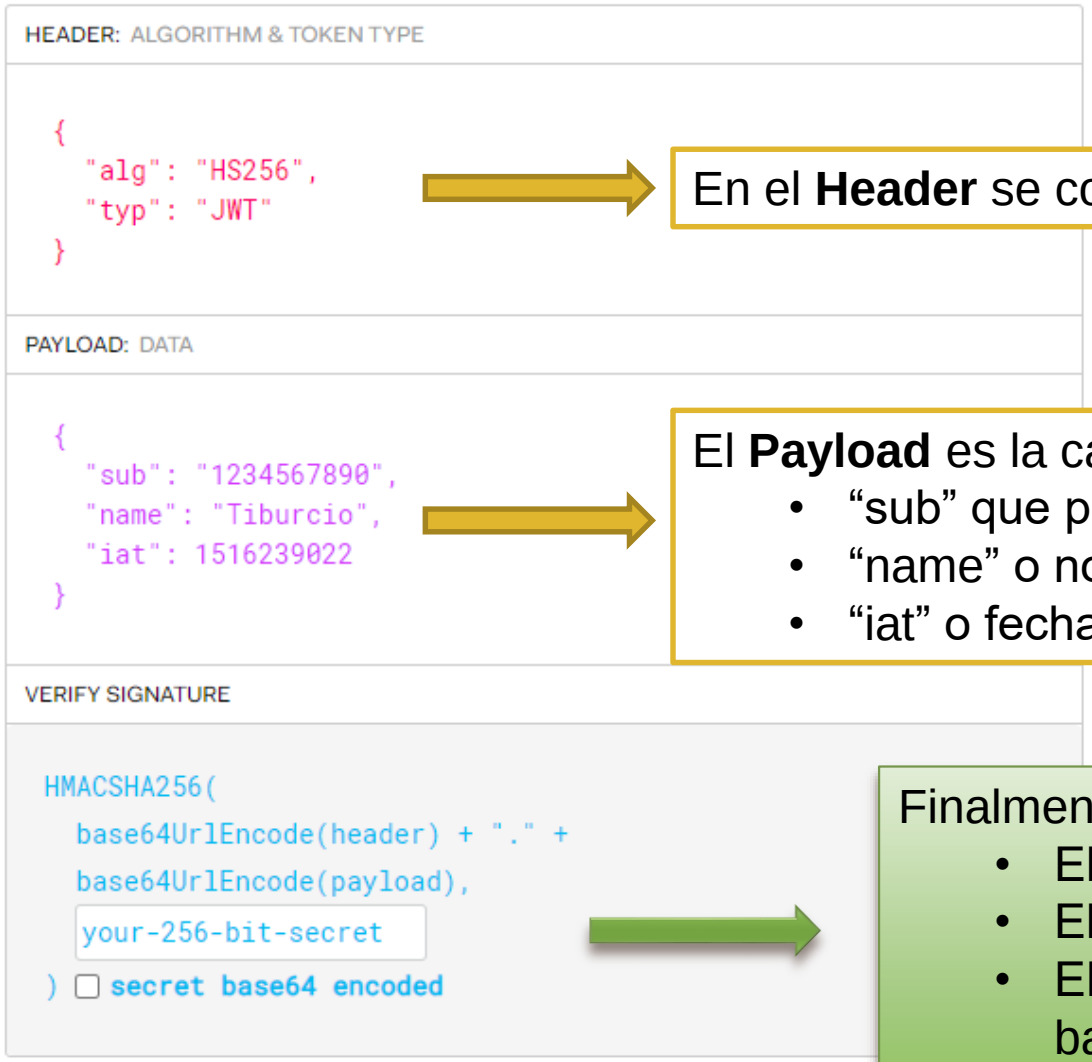


Visión de conjunto usando **Basic auth** y **Token bearer** conjuntamente

JWT – Json Web Token (https://jwt.io/)



JWT – Json Web Token (formato de Token muy utilizado)



En el **Header** se coloca el algoritmo y tipo de token

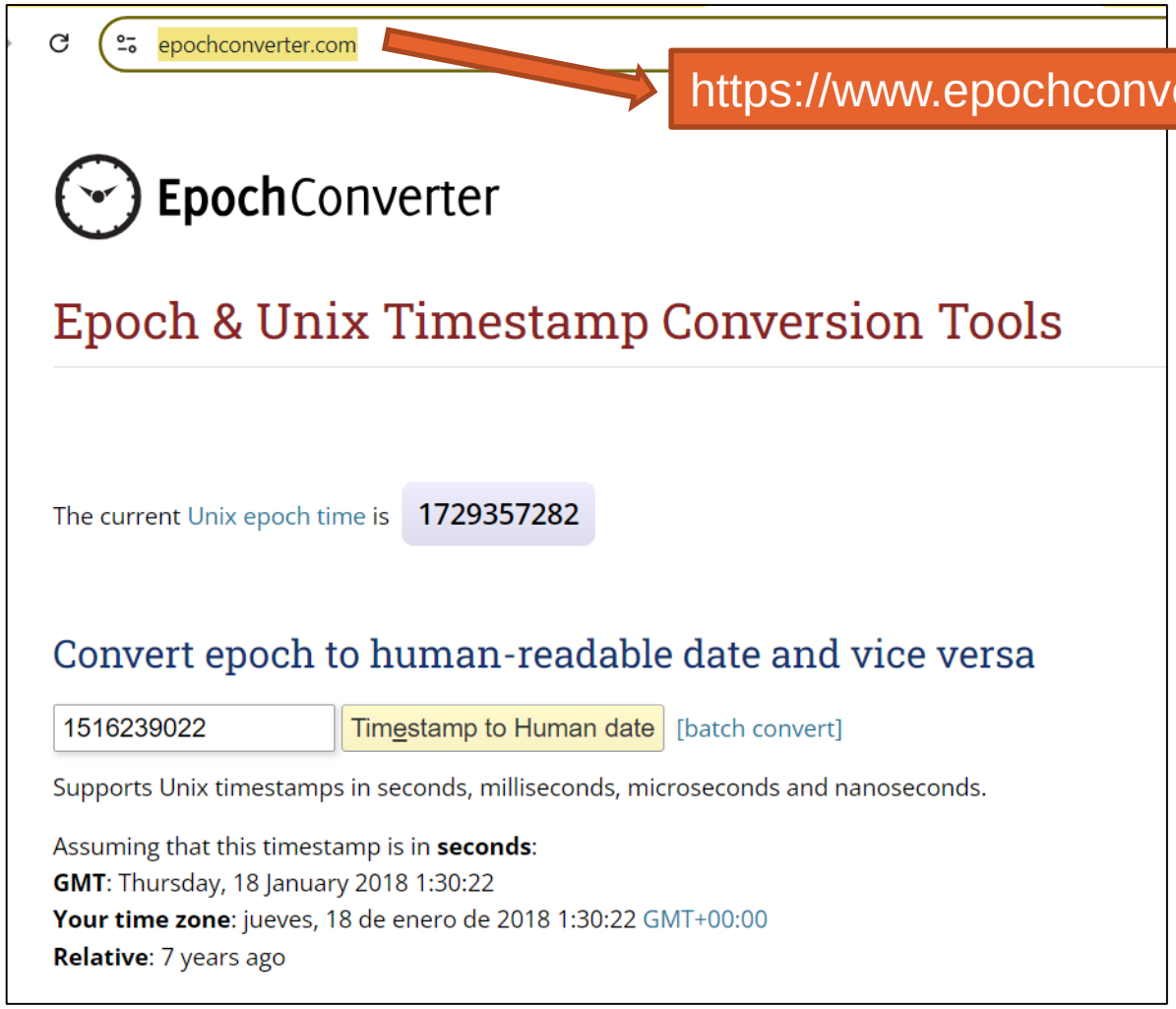
El **Payload** es la carga útil. Por ejemplo:

- "sub" que puede ser el id del usuario.
- "name" o nombre del usuario.
- "iat" o fecha del token

Finalmente el **token JWT** es:

- El "**header**" codificado en base64
- El "**Payload**" codificado en base64
- El "**secreto**" que conoce sólo el backend.

Epoch Converter
(Sitio web para convertir entre Unix Timestamp y formato legible por seres humanos)



Unix Timestamp, es el número de segundos, microsegundos o nanosegundos (según queramos) desde el 1 de enero de 1970.

Formato muy usado como ***timestamp***, para marcar el momento en el que ocurre algún evento.

Se usa por ejemplo para el campo **iat** del token **JWT**

¿Es seguro usar tokens?

Usar un **token** una vez se ha realizado la **autenticación básica** es más seguro que solo usar autenticación básica.

¿Es seguro usar tokens?

Usar un **token** una vez se ha realizado la **autenticación básica** es más seguro que solo usar autenticación básica.

Esto es así porque:

- *El token tiene una vida limitada.*
- *El secreto sólo lo conoce el backend que puede verificar si el token es válido.*

¿Es seguro usar tokens?

Usar un **token** una vez se ha realizado la **autenticación básica** es más seguro que solo usar autenticación básica.

Esto es así porque:

- *El token tiene una vida limitada.*
- *El secreto sólo lo conoce el backend que puede verificar si el token es válido.*

Sin embargo un **token** se puede analizar, por ejemplo, en la web ***https://jwt.io/***

Y un atacante podría usarlo si lo consigue de alguna manera

¿Es seguro usar tokens?

Usar un **token** una vez se ha realizado la **autenticación básica** es más seguro que solo usar autenticación básica.

Esto es así porque:

- *El token tiene una vida limitada.*
- *El secreto sólo lo conoce el backend que puede verificar si el token es válido.*

Sin embargo un **token** se puede analizar, por ejemplo, en la web ***https://jwt.io/***

Y un atacante podría usarlo si lo consigue de alguna manera

Entonces… ¿Cuál es la solución?

¿Es seguro usar tokens?

Usar un **token** una vez se ha realizado la **autenticación básica** es más seguro que solo usar autenticación básica.

Esto es así porque:

- *El token tiene una vida limitada.*
- *El secreto sólo lo conoce el backend que puede verificar si el token es válido.*

Sin embargo un **token** se puede analizar, por ejemplo, en la web ***https://jwt.io/***

Y un atacante podría usarlo si lo consigue de alguna manera

Entonces… ¿Cuál es la solución? → ***En cualquier caso hay que usar un certificado SSL (https)***

Y ahora veamos un ejemplo probando con POSTMAN

AV
AV
AVeryStrongPasswordSoThatNobodyCouldBreakIt

(sigue las instrucciones del README.md)

Clona el siguiente proyecto y sigue las instrucciones de su README.md:
https://github.com/tcrurav/Ionic8NodeAuthBasic

En las instrucciones del ***README.md*** verás que habla de un ***fichero .env***

Un fichero ***.env*** contiene la configuración particular de la instalación donde se va a ejecutar el proyecto. Resulta muy interesante para publicar un proyecto y poner los detalles que cambian que son propios de una instalación concreta como las credenciales de la BD o el secreto del JWT.

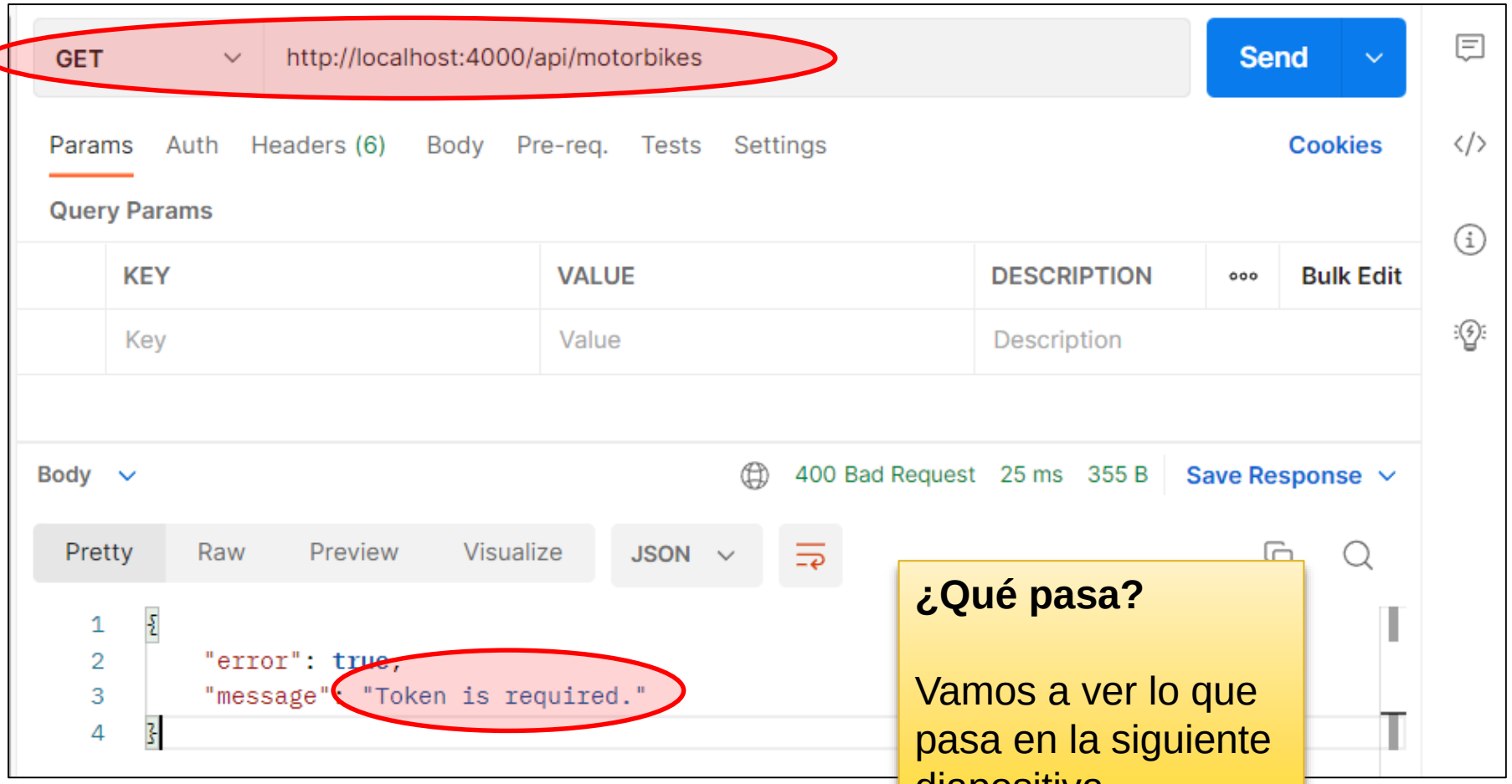
Ejemplo de **.env** →

```
JWT_SECRET=AVeryStrongPassword
MYSQL_DATABASE=db_motorbikes_dev
MYSQL_USER=root
MYSQL_PASSWORD=sasa
MYSQL_ROOT_PASSWORD=sasa
DB_HOST=localhost
NODE_ENV=development
```

Una vez esté arrancado el backend del proyecto ya podemos probarlo desde POSTMAN

(para esta parte no te hace falta el frontend porque probaremos con POSTMAN)

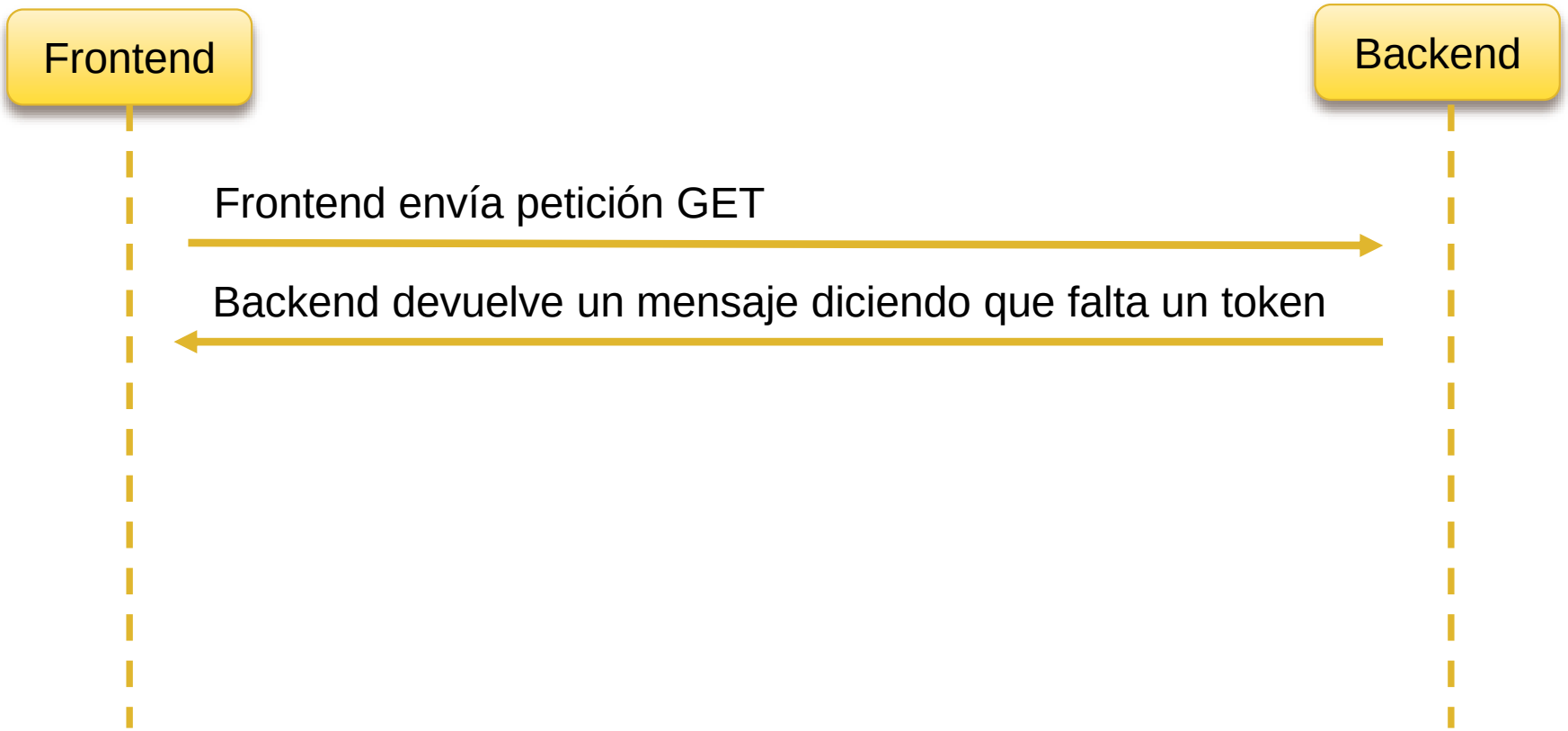
GET no funciona
(Se requiere un token)



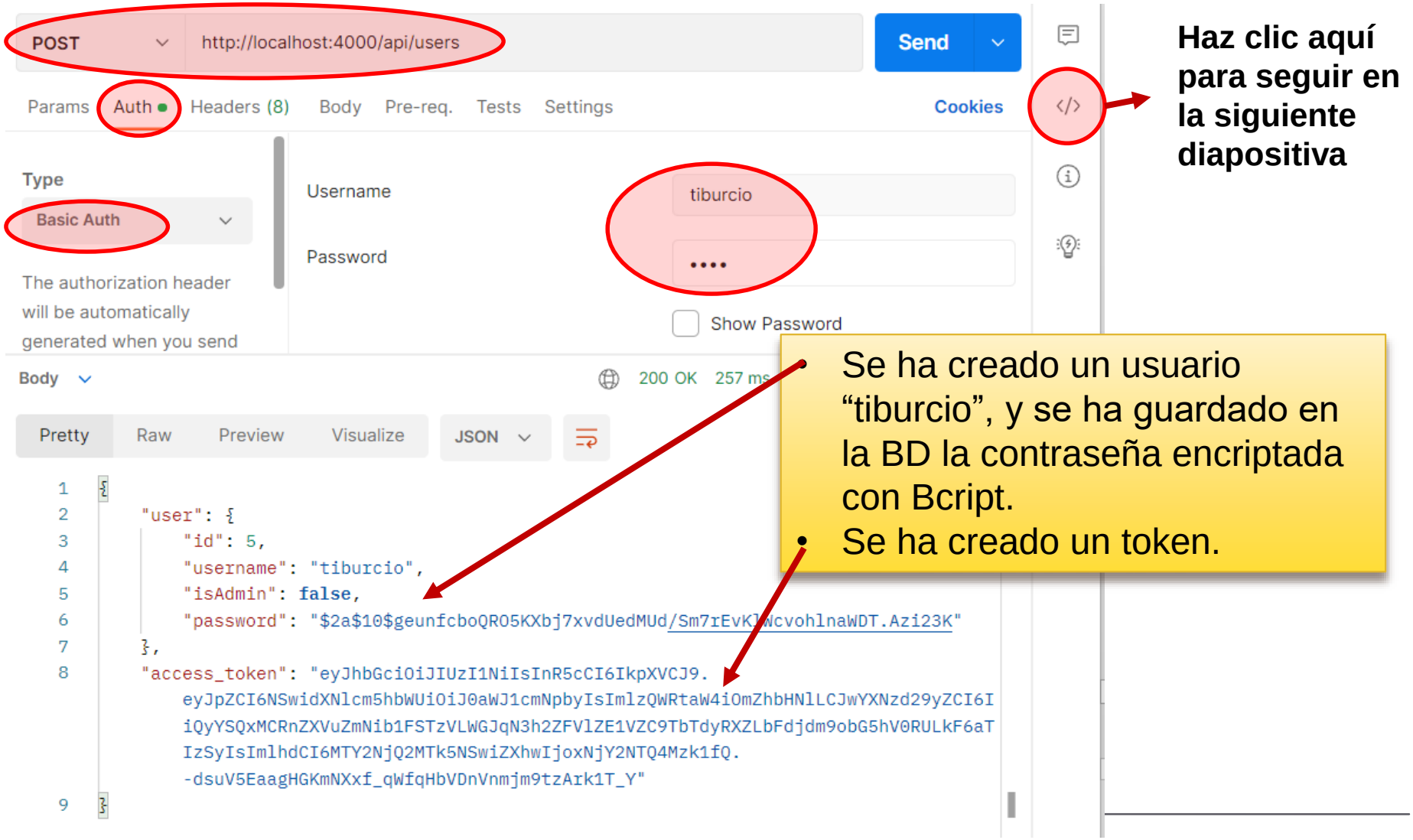
¿Qué pasa?

Vamos a ver lo que pasa en la siguiente diapositiva.

Esto es lo que hemos hecho...



Primero hay que registrarse (Ahora ya tenemos el token)



Haz clic aquí para seguir en la siguiente diapositiva

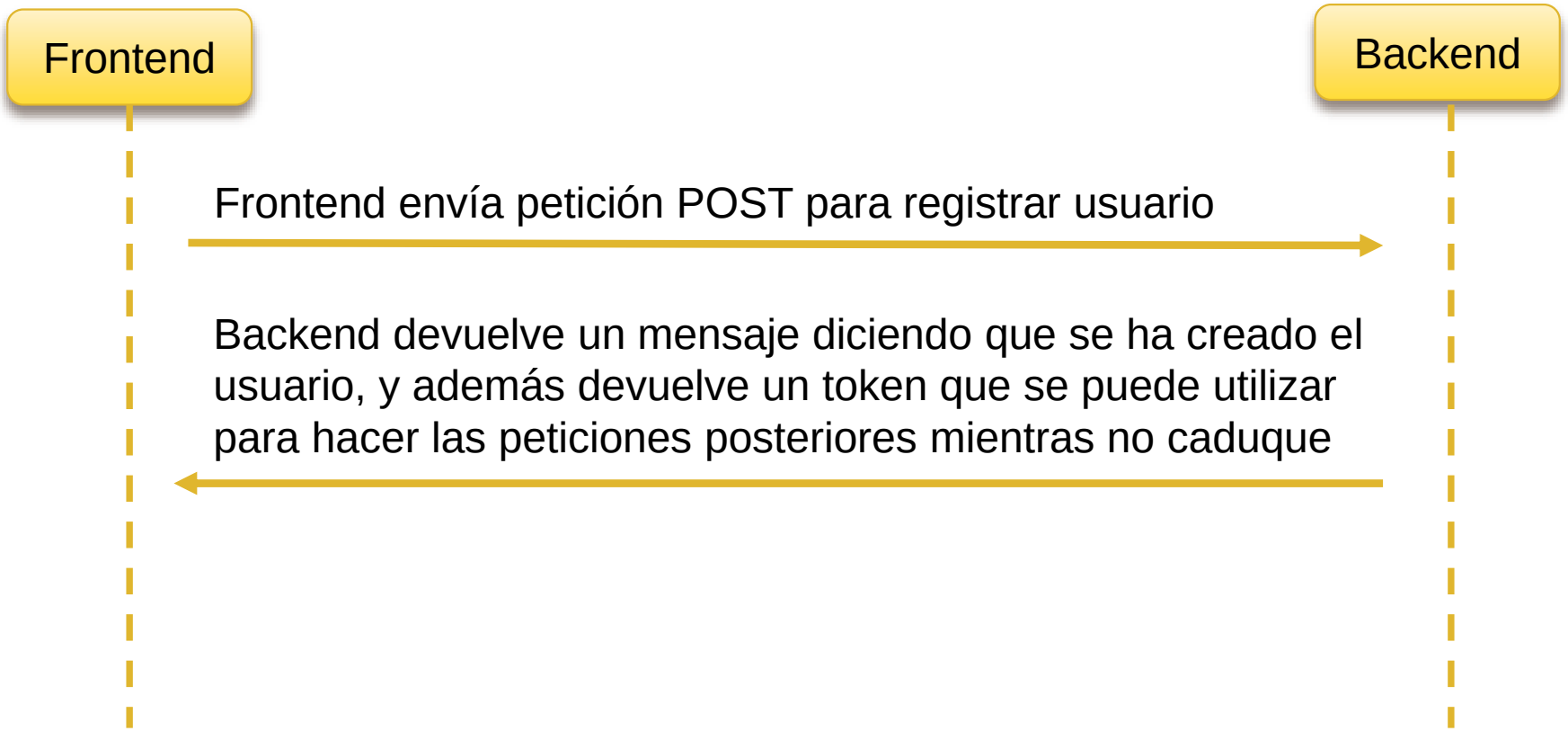
- Se ha creado un usuario “tiburcio”, y se ha guardado en la BD la contraseña encriptada con Bcript.
- Se ha creado un token.

POSTMAN muestra cómo hacer la petición con variedad de librerías/lenguajes
(En este caso con axios)

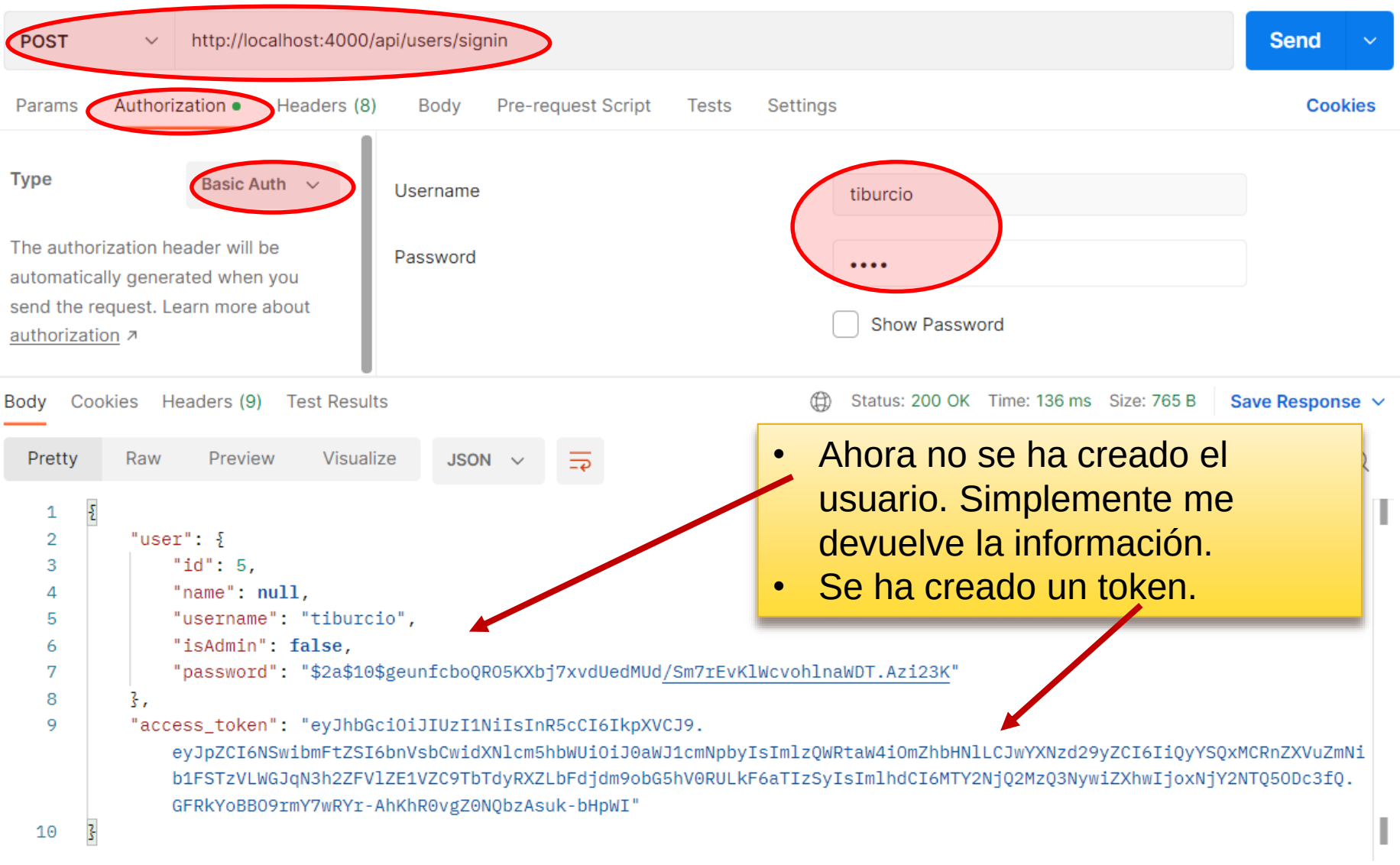
Al haber hecho clic aquí ahora puedes ver cómo sería la llamada usando la librería axios desde un frontend.

Observa que estamos usando Autorización básica

Esto es lo que hemos hecho...

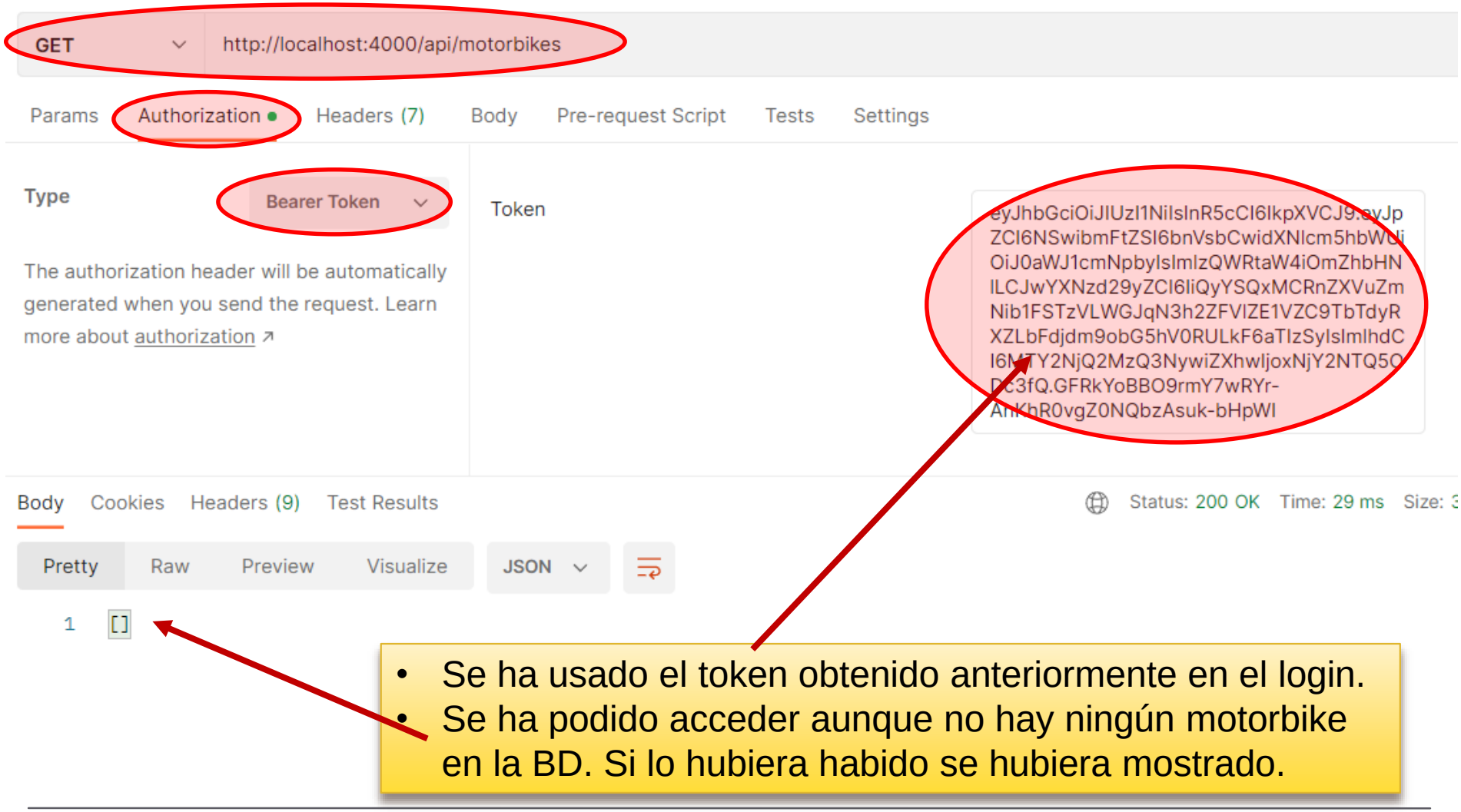


Ahora podemos hacer un login
(porque ya el usuario “tiburcio” está registrado)



- Ahora no se ha creado el usuario. Simplemente me devuelve la información.
- Se ha creado un token.

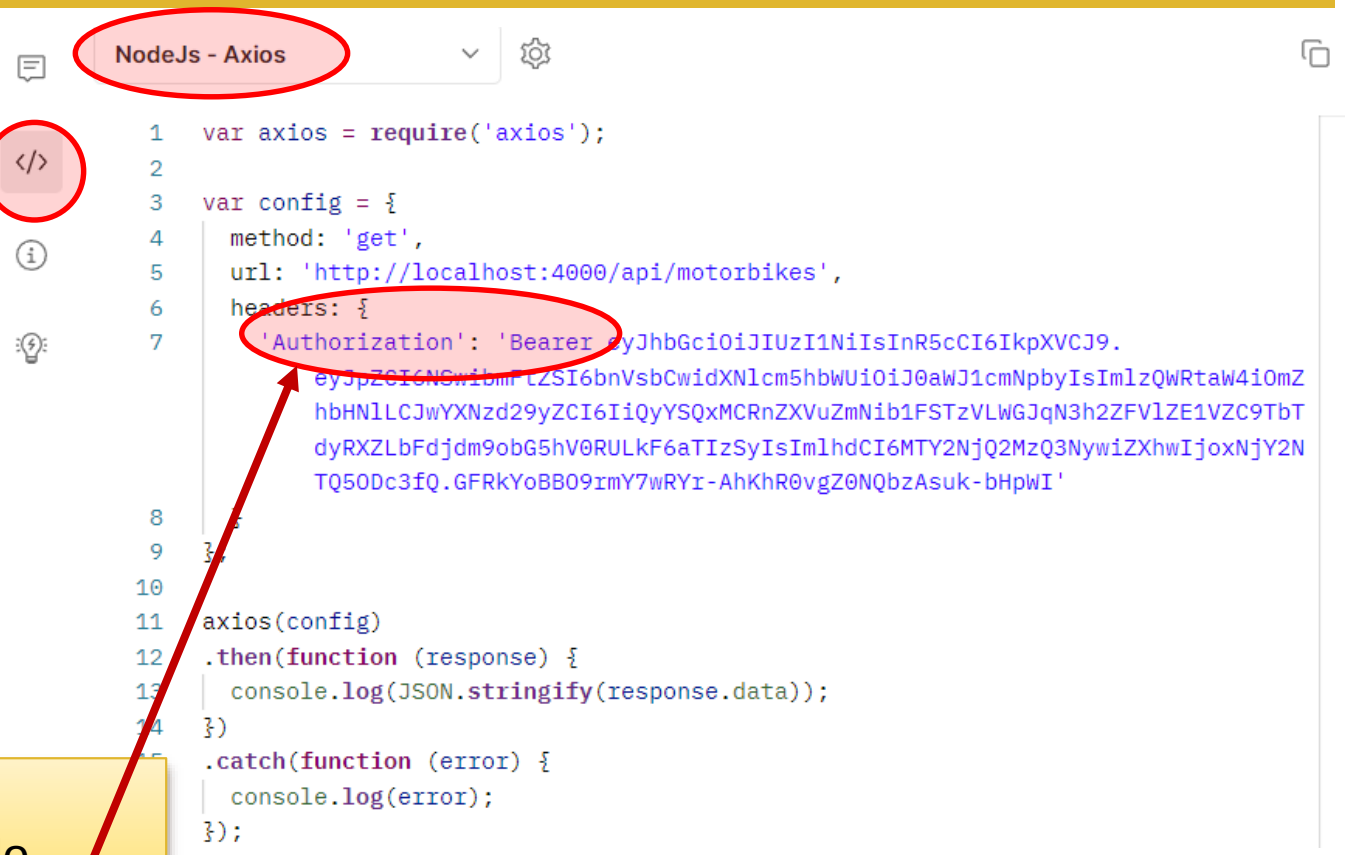
Con el token del login ya podemos acceder al listado de motorbikes al que antes no se podía



- Se ha usado el token obtenido anteriormente en el login.
- Se ha podido acceder aunque no hay ningún motorbike en la BD. Si lo hubiera habido se hubiera mostrado.

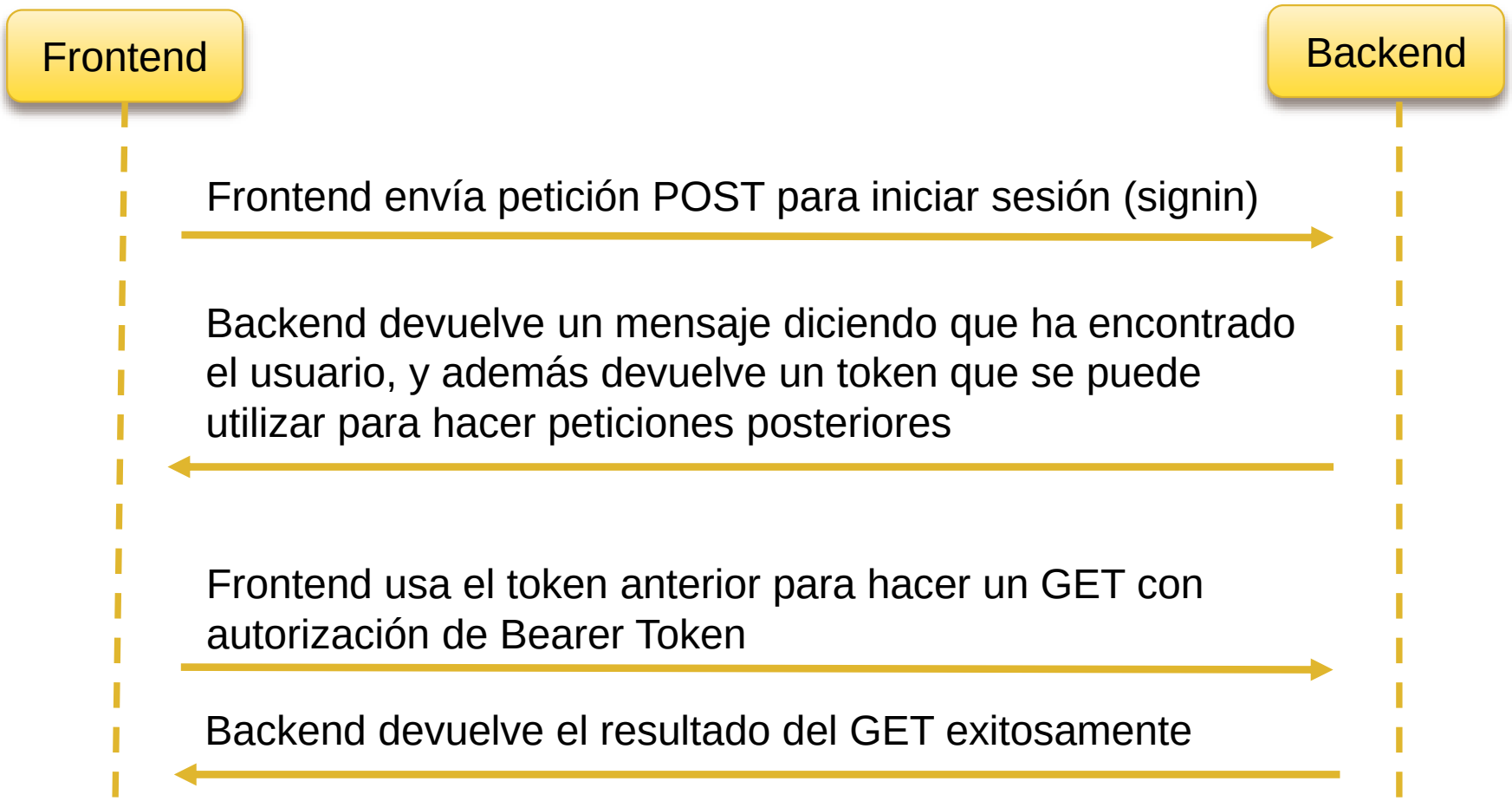
Observa cómo sería la petición usando la librería axios

Al hacer clic aquí ahora puedes ver cómo sería la llamada usando la librería axios desde un frontend.



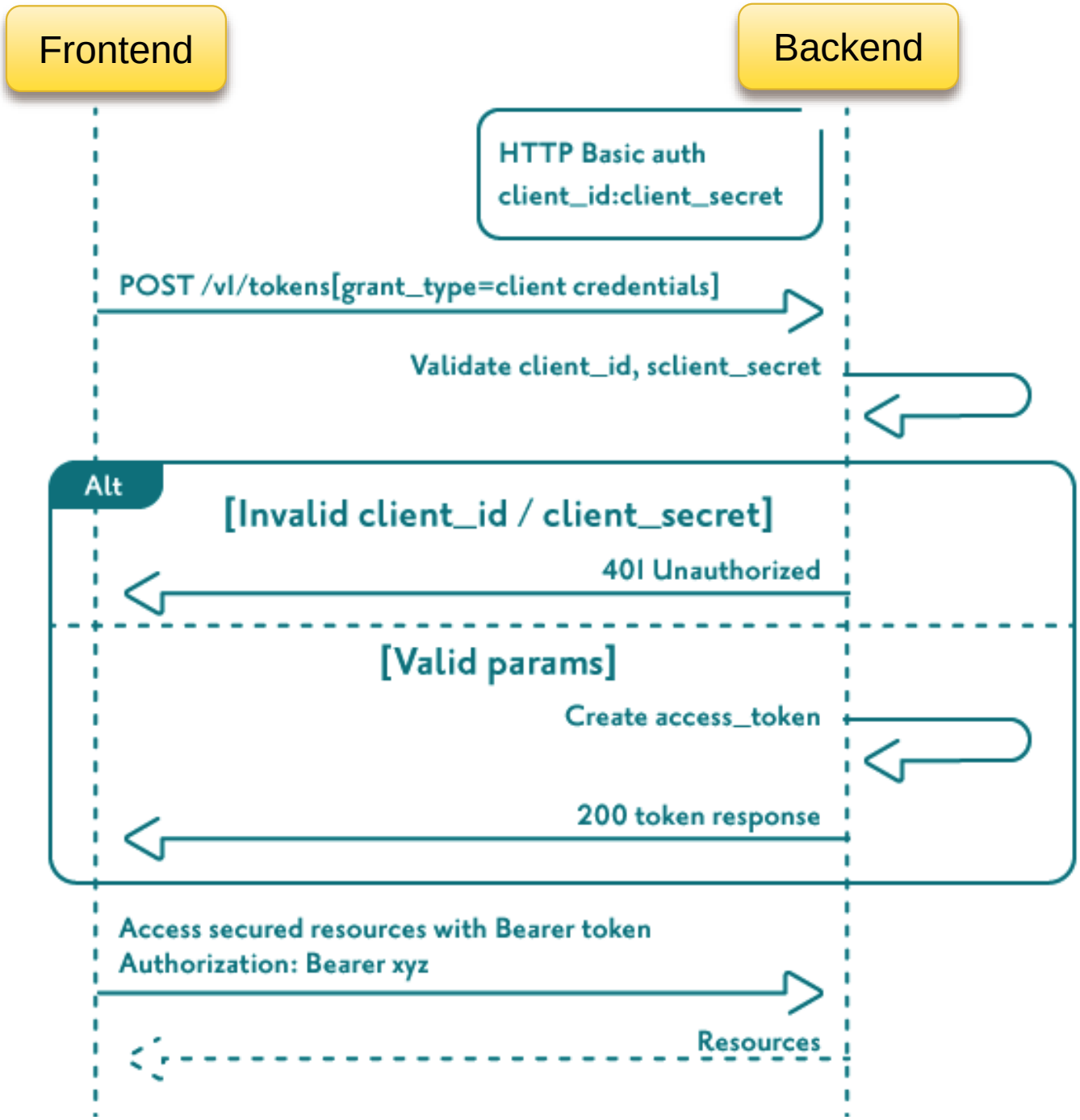
Observa que estamos usando Autorización con Bearer token

Esto es lo que hemos hecho...



Y con eso ya hemos conseguido probar el proyecto con POSTMAN

(Vamos a verlo de nuevo de forma esquemática en la siguiente diapositiva)



Visión de conjunto usando **Basic auth** y **Token bearer**

A continuación se comentan algunos de los detalles más relevantes del proyecto que tienen que ver con la autenticación

En backend/server.js al recibir:
'Authorization': 'Basic dGlidXJjaW86MTIzNA==‘

```
app.use(function (req, res, next) {
  // check header or url parameters or post parameters for token
  var token = req.headers['authorization'];
  if (!token) return next(); //if no token, continue

  if(req.headers.authorization.indexOf('Basic ') === 0){
    // verify auth basic credentials
    const base64Credentials =  req.headers.authorization.split(' ')[1];
    const credentials = Buffer.from(base64Credentials, 'base64').toString('ascii');
    const [username, password] = credentials.split(':');

    req.body.username = username;
    req.body.password = password;

    return next();
  }
```

Este fragmento de código comprueba que llega esta cabecera de autenticación básica, descodifica el base64, y extrae el usuario y contraseña. Finalmente al retornar “next()” sigue con el siguiente paso de tratamiento de la petición.

En backend/server.js al recibir:
'Authorization': 'Bearer dGlidXJjaW86MT...'

```
  token = token.replace('Bearer ', '');
  // .env should contain a line like JWT_SECRET=V3RY#1MP0RT@NT$3CR3T#
  jwt.verify(token, process.env.JWT_SECRET, function (err, user) {
    if (err) {
      return res.status(401).json({
        error: true,
        message: "Invalid user."
      });
    } else {
      req.user = user; //set the user to req so other routes can use it
      req.token = token;
      next();
    }
  });
});
```

Este fragmento de código comprueba que llega una cabecera con un token bearer, usa el "secreto" para comprobar que efectivamente el token fue codificado previamente con dicho "secreto", y extrae el usuario y el token. Finalmente al ejecutar "next()" sigue con el siguiente paso de tratamiento de la petición.

En backend/routes/user.routes.js

```
// Create a new User
router.post("/", users.create);

// Retrieve all User
router.get("/", auth.isAuthenticated, users.findAll);

// Retrieve a single User with id
router.get("/:id", auth.isAuthenticated, users.findOne);

// Update a User with id
router.put("/:id", auth.isAuthenticated, users.update);

// Sign in
router.post("/signin", auth.signin);
```

En este fragmento de código se observa que las rutas para el login y el registro no llaman al middleware **auth.isAuthenticated**, porque para acceder a dichas rutas no es necesario estar autentificado

El resto de rutas sí están protegidas, porque sólo si el usuario está autentificado se puede acceder al método del controlador correspondiente. Por ejemplo sólo se puede listar los usuarios (users.findAll) si se está autentificado.

En backend/controllers/auth.js

El método **auth.isAuthenticated**, verifica que el token es válido, y que el usuario existe en la BD.

Si se cumple lo anterior significa que el usuario que está usando el token se autentificó correctamente anteriormente, y por tanto puede acceder al recurso.

```
exports.isAuthenticated = (req, res, next) => {
  var token = req.token;
  if (!token) {
    return res.status(400).json({
      error: true,
      message: "Token is required."
    });
  }
  // check token that was passed by decoding token using secret
  // .env should contain a line like JWT_SECRET=V3RY#1MP0RT@NT$3CR3T#
  jwt.verify(token, process.env.JWT_SECRET, function (err, user) {
    if (err) return res.status(401).json({error: true, message: "Invalid token."});

    User.findByPk(user.id)
      .then(data => {
        // return 401 status if the userId does not match.
        if (!user.id) {
          return res.status(401).json({error: true,message: "Invalid user."});
        }
        // get basic user details
        next();
      })
      .catch(err => {
        res.status(500).send({message: "Error retrieving User with id=" + id});
      });
```

En backend/controllers/auth.js

```
exports.signin = (req, res) => {
  const user = req.body.username;
  const pwd = req.body.password;

  // return 400 status if username/password is not exist
  if (!user || !pwd) {
    return res.status(400).json({error: true,message: "Username or Password required."});
  }

  // return 401 status if the credential is not match.
  User.findOne({ where: { username: user } })
    .then(data => {
      const result = bcrypt.compareSync(pwd, data.password);
      if(!result) return  res.status(401).send('Password not valid!');

      // generate token
      const token = utils.generateToken(data);
      // get basic user details
      const userObj = utils.getCleanUser(data);
      // return the token along with user details
      return res.json({ user: userObj, access_token: token });
    })
    .catch(err => {
      res.status(500).send({
        message: err.message || "Some error occurred while retrieving tutorials."
      });
    });
};
```

El método **auth.signin**, recibe un usuario y contraseña del usuario que quiere hacer el login.

Busca el usuario en la BD. Si lo encuentra ejecuta **bcrypt.compareSync** para comprobar que la clave encriptada en la BD coincide con la encriptación de la contraseña pasada por el usuario. Si es así genera un token para el usuario.

En backend/controllers/user.controller.js

El método **create** es dónde se hace efectivo el registro de un nuevo usuario.

```
user.password = bcrypt.hashSync(req.body.password);

// User not found. Save new User in the database
User.create(user)
  .then(data => {
    const token = utils.generateToken(data);
    // get basic user details
    const userObj = utils.getCleanUser(data);
    // return the token along with user details
    return res.json({ user: userObj, access_token: token });
  })
  .catch(err => {
    res.status(500).send({
      message:
        err.message || "Some error occurred while creating the User."
    });
  });

})
```

Al ejecutar **bcrypt.hashSync** se encripta la contraseña del usuario para almacenarla en la BD.

En backend/utils.js

```
var jwt = require('jsonwebtoken');

// generate token and return it
function generateToken(user) {
  //1. Don't use password and other sensitive fields
  //2. Use the information that are useful in other parts
  if (!user) return null;

  var u = {
    id: user.id,
    name: user.name,
    username: user.username,
    isAdmin: user.isAdmin,
    password: user.password
  };

  // .env should contain a line like JWT_SECRET=V3RY#1MP0RT@NT$3CR3T#
  return jwt.sign(u, process.env.JWT_SECRET, {
    expiresIn: 60 * 60 * 24 // expires in 24 hours
  });
}
```

El método **generateToken** es dónde se crea un token, que permitirá al usuario trabajar con él mientras no caduque.

Al ejecutar **jwt.sign** se está usando el “secreto” que el backend usa para codificar el token, y asegurar que dicho token no ha sido manipulado.

En frontend/src/app/auth/auth.service.ts

```
AUTH_SERVER_ADDRESS:  string  =  'http://localhost:4000';

constructor(private  httpClient:  HttpClient, private  storage:  Storage) { }

private getOptions(user: User){
  let base64UserAndPassword = window.btoa(user.username + ":" + user.password);

  let basicAccess = 'Basic ' + base64UserAndPassword;

  let options = {
    headers: {
      'Authorization' : basicAccess,
      'Content-Type' : 'application/x-www-form-urlencoded',
    }
    //, withCredentials: true
  };

  return options;
}
```

El método **getOptions** crea la cabecera en la que usuario:contraseña irá codificada en base64.

El método window.btoa es el que realiza la codificación en base64.

En frontend/src/app/auth/auth.service.ts

```
register(user: User): Observable<AuthResponse> {
  return this.httpClient.post<AuthResponse>(
    `${this.AUTH_SERVER_ADDRESS}/api/users/`, user, this.getOptions(user)
    ).pipe(tap(async (res:  AuthResponse ) => {
      if (res.user) {
        await this.storage.set("token", res.access_token);
      }
    })

  );
}
```

El método **register** usa la cabecera en la que usuario:contraseña va codificada en base64 que crea el método **getOptions**. El método **register** hace una petición al backend con dicha cabecera para registrar al usuario.

El método **storage.set** guarda el token en el **LocalStorage** para poderlo utilizar en otras peticiones. El **LocalStorage** es una memoria del navegador que no se borra incluso aunque se cierre el navegador.

En frontend/src/app/auth/auth.service.ts

```
login(user: User): Observable<AuthResponse> {
  return this.httpClient.post(
    `${this.AUTH_SERVER_ADDRESS}/api/users/signin`, null, this.getOptions(user)
    ).pipe(tap(async (res: AuthResponse) => {
      if (res.user) {
        await this.storage.set("token", res.access_token);
      }
    })
  );
}
```

El método **login** usa la cabecera en la que usuario:contraseña va codificada en base64 que crea el método **getOptions**. El método **login** hace una petición al backend con dicha cabecera para iniciar sesión. Si todo va bien el backend devolverá el token.

El método **storage.set** guarda el token en el **LocalStorage** para poderlo utilizar en otras peticiones. El **LocalStorage** es una memoria del navegador que no se borra incluso aunque se cierre el navegador.

En frontend/src/app/auth/auth.service.ts

```
async logout() {
  await this.storage.remove("token");
}

async isLoggedIn() {
  let token = await this.storage.get("token");
  if (token){ //Just check if exists. This should be checked with current date
    return true;
  }
  return false;
}
```

El método **logout** simplemente borra el token del **LocalStorage**, de manera que la próxima vez que se quiera enviar una petición no se va a encontrar el token que es equivalente a no estar logueado.

El método **isLoggedIn** lee el token del **LocalStorage.** Si no está significa que el usuario no puede usar el token y por tanto es equivalente a no estar logueado.

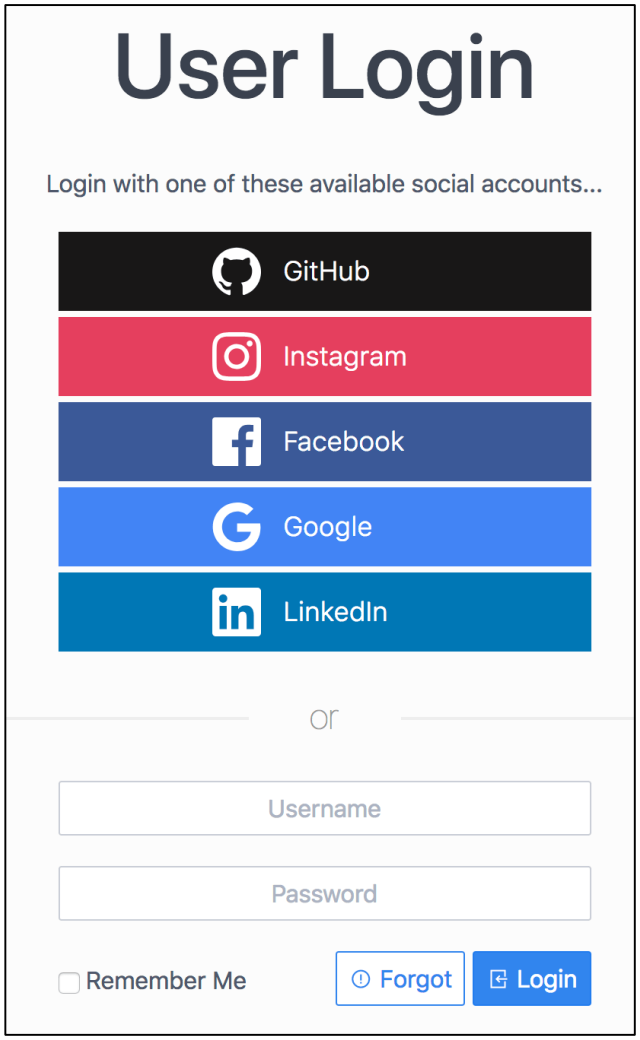
En frontend/src/app/services tanto motorbike.service.ts como user.service.ts

```
AUTH_SERVER_ADDRESS:  string  =  'http://localhost:4000';

constructor(private  httpClient:  HttpClient, private  storage:  Storage) { }

private getOptions(token){

  let bearerAccess = 'Bearer ' + token;

  let options = {
    headers: {
      'Authorization' : bearerAccess,
    }
  };

  return options;
}
```

El método **getOptions** prepara la cabecera con autorización de Bearer Token para que las peticiones la incluyan y por tanto puedan demostrar que se han autentificado.

Y de esta manera se han comentado las partes más relevantes del código del proyecto de ejemplo que tienen que ver con la autentificación

Sigue aprendiendo...



El siguiente paso en la autenticación es usar **OAuth2**

Se trata de delegar en un tercero la autenticación.

Tiene como ventaja para el usuario la facilidad de no tener que tener diferentes credenciales para cada servicio al que quiere acceder.

Conclusiones

¿Qué hemos aprendido?

- En qué consiste la autenticación básica.
- Cómo se usa base64 y una utilidad online que nos permite codificar/descodificar.
- Cómo se usa Bcript, y una utilidad online que nos permite encriptar y verificar información encriptada.
- Que es necesario usar Https para que la autenticación sea segura.
- Hemos aprendido qué es un token, y el formato JWT de token.
- También aprendimos una utilidad online para analizar tokens JWT.
- Hemos probado el proyecto con POSTMAN.
- Hemos comentado las partes más importantes del código del proyecto de ejemplo relacionadas con la autenticación.
- Ahora sabemos qué es OAuth2.

Próximos pasos…

•Prueba el ejemplo de cómo usar la autenticación básica y tokens JWT con Ionic (frontend) y Express + Sequelize + MySQL en el backend, que tienes en https://github.com/tcrurav/Ionic8NodeAuthBasic