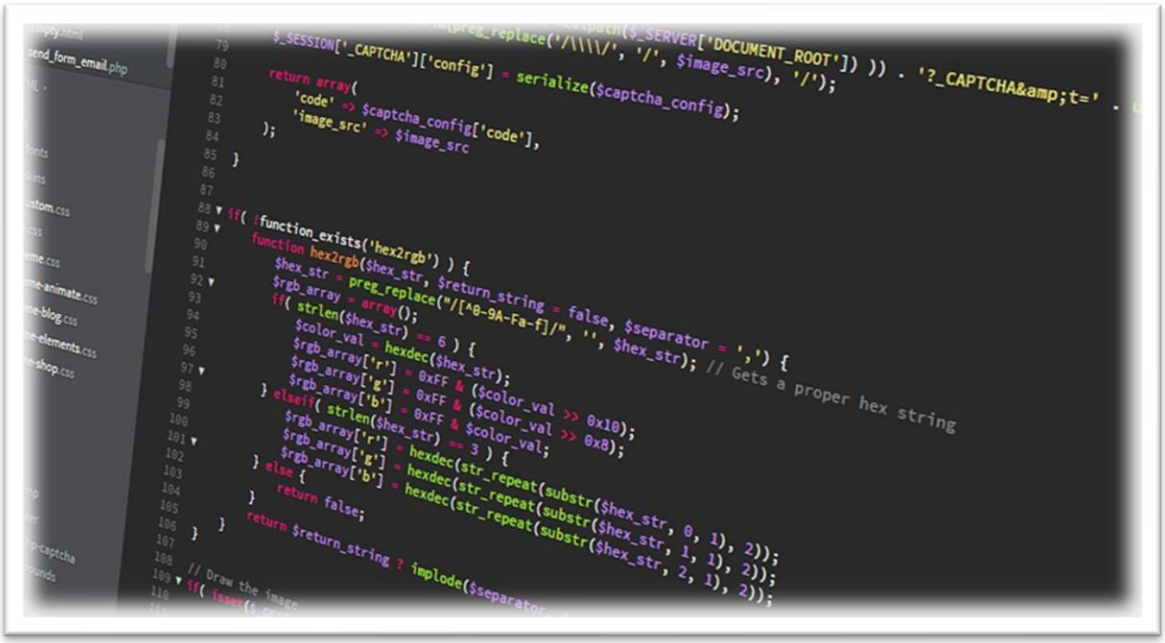
Manual de uso para el backend

Aplicación de recompensas



Índice de contenidos

Resumen

El desarrollo de este backend tiene como fin la construcción de una plataforma, que con las tecnologías Web-Services y Blockchain, permita a los usuarios del simulador iVictrix, obtener puntos y recompensas por el desempeño en las sesiones de entrenamiento de este.

La parte Blockchain almacenará las diferentes transacciones que haya entre los usuarios de la plataforma, dotando de seguridad a los cambios no intencionados gracias a los hashes y los Web-Services se encargarán de atender a las diferentes peticiones que hagan los usuarios a esta plataforma.

Remarcar que los usuarios y las recompensas y puntos no tienen ninguna conectividad con el simulador iVictrix. En un futuro, si se desea estudiar una implementación a dicha plataforma, se puede transformar el backend para ello. Actualmente no se recoge esta idea en el backend.

Para los diferentes elementos de la plataforma, se usan los algoritmos de encriptación como SHA-256 para dar propiedades únicas a estos y también, dicho algoritmo es usado en los hashes de los bloques de la Blockchain para darles encriptación a la información de su interior.

1. Elementos destacables

En este punto se procede a explicar elementos destacables del backend, para su compresión y entendimiento.

1.1 Clases de los objetos

Con la creación de las clases de los objetos, se consiga dar funcionalidad dentro del backend a través de las instancias de estos. Las clases implementadas son las siguientes:

1.1.1 Block

La clase Block se encarga de correcta creación de los bloques para la blockchain. Los atributos de esta clase son: index, hash, timestamp, transaction y hashPrev. Index es un identificador para el bloque. Hash y hashPrev son donde se almacenan la información cifrada del bloque actual y del bloque inmediatamente anterior.

Timestamp es una marca de tiempo que es usada para dejar constancia del momento de la creación del bloque. Transaction almacena todos los ids de las transacciones que se han creado durante la función cíclica (Su correspondiente explicación se encuentra en el apartado X).

Toda esta información es rellenada por medio de un constructor en la propia clase.

Por último, esta clase tiene el siguiente método:

```
calculateHash() {
    return SHA256(this.index + this.timestamp + +JSON.stringify(this.transactionIds) + this.hashPrev).toString();
}
```

Esta clase se encarga de generar los hashes a través del algoritmo SHA-256. Gracias a Crypto.js, basta con hacer un require de la librería, en la clase y llamar a esa variable y pasarle la información a encriptar.

1.1.2 Wallet

La clase Wallet se encarga de correcta creación de los monederos para la plataforma. Los atributos de esta clase son: owner, keyPublic y keyPrivate.

Owner es el identificador del usuario al que pertenece el monedero, keyPublic y keyPrivate son las claves del monedero y son usadas como identificador para las transacciones y como firma respectivamente. Estas claves son generadas por la librería elliptic. Esta librería, al igual que crypto js, sirve para encriptar información, pero también esta su uso muy estandarizado en la generación de claves para los monederos y carteras virtuales. Esta clase tiene el siguiente método:

```
generatePublicPrivateKey(){
    const key = ec.genKeyPair();
    this.keyPublic = key.getPublic('hex');
    this.keyPrivate = key.getPrivate('hex');
}
```

Este método se encarga de generar las claves para el monedero. El algoritmo a grandes rasgos es obtener puntos de una curva elíptica, obtener 3 puntos concretos de ella y generar las claves a partir de estos valores.

El comando ec.genKeyPair, se genera una cadena alfanumérica de la cual si se llama después a los métodos getPublic y getPrivate se obtienen las claves. Al usar el algoritmo de encriptado SHA-256 aquí también, se limita el número de caracteres de la cadena alfanumérica a 64 bits. Ese valor es guardado en key y con este mismo, se generan las claves privada y pública a través de los métodos que se muestran en la Figura. Remarcar que con elliptic, se pueden obtener la privateKey con la publicKey y viceversa.

1.1.3 UniReward

Esta clase se encarga de crear el objeto UniReward. Esta clase tiene los siguientes atributos: id, nameUR, descriptionUR, imageUR, cost, moneyExp, purchase, hash y WalletId. Hash es el campo que permite almacenar el hash formado por la transacción en la que se creó la UniReward y el id de la misma. Esto sirve para hacer única a la UniReward. Este valor se repite en la clase UniPoints y en la clase Transactions. Los atributos id, nameUr, descriptionUR, imageUR, cost, moneyExp, purchase y WalletId, hacen referencia al nombre, a la descripción, a la URL de la imagen, al coste de la UniReward, a la cantidad de UniPoints ingresados de la UniReward, al estado de la UniReward y al dueño de la UniReward.

Como funciones, se tienen las siguientes:

```
proveNotNullObject() {
    if (this.nameUR != null && this.descriptionUR != null && this.imageUR != null && this.cost != null  && this.WalletId != null) {
        return false
    } else {

        return true
    }
}

async getAndSetLastId() {
    var lastID = await controllerDB.getLastUniRewardIndex()
    this.id = lastID
}
```

La primera función se encarga que los datos de la clase estén correctamente rellenados. La segunda función se encarga de buscar el id que va a tener la UniReward en base de datos.

1.1.4 Transaction

Esta clase se encarga de la creación del objeto para las transacciones de la plataforma. Como atributos de esta, se tienen: El id de la transacción (id), las direcciones de destinatario y remitente (fromAddress y toAddress respectivamente), timestamp, que es una marca de tiempo que es usada para saber el momento de la creación de la transacción. Concept, es el concepto de la transacción, al igual que en transacciones con el banco a la hora de un pago. Las firmas de destinatario y remitente (signatureFrom y signatureTo respectivamente), para dejar constancia del consentimiento de los usuarios. La id de la UniReward sobre la que gira la transacción. Amout es la cantidad de UniPoints que se desea traspasar. Las ids de las transacciones tanto del usuario (idWalletFrom y idWalletTo), el tipo de transacción (typeT) y las ids de los UniPoints a traspasar uniPointsIds.

Como funciones de esta clase, se tienen:

La función getAndSetLastTransactionId se encarga de buscar y asignar un id para la instancia de la clase. Se encarga de buscar en la base de datos la última id que se puede asignar a dicha instancia. La función setUniPointIds, se encarga de asignar las ids de los UniPoints que se usan en la transacción y actualizarlos en el objeto.

Las tres últimas funciones se encargan de crear las firmas para las transacciones. La función calHashTransaction se encarga de generar parte de la firma, generando un hash con los datos del constructo de la clase. SingTransaction se encarga de firmar la transacción, esta función está pensada para poder firmar tanto como si se es el remitente como si se es el destinatario. Esta diferenciación se obtiene por uno de los parámetros de la función, type. Con este valor si es igual a 0 firma remitente y si no es así, como destinatario.

A esta función se le pasa la clave privada del monedero y el tipo de firma que se espera. El sistema pasa la clave privada de string a clave privada, comprueba el tipo de firma y comprueba si con la privateKey se puede obtener la publicKey del usuario en concreto, si es así, generar todo lo necesario para crear la firma. En caso negativo, no se rellenan eso campos.

Por último, isValid se encarga de comprobar la validez de las firmas anteriormente creadas. Lo único que hace falta saber sobre esta función es que en caso de que alguno de los datos no sea correcto, se devuelve que no es correcta, en caso afirmativo, devolverá que son correctas.

1.1.5 SmartContract

Esta clase se encarga de la creación del objeto para los SmartContracts de la plataforma. Son los encargados de monitorizar las transacciones alrededor de una UniReward, desde su creación, hasta su finalización y entrega. Como atributos de esta, se tienen: Las direcciones de destinatario y remitente (walletIdObserver y walletIdDemander respectivamente). Las firmas de destinatario y remitente (signatureObserver y signatureDemander) respectivamente), para dejar constancia del consentimiento de los usuarios. El estado o state es la variable que se encarga de ver el estado del contrato, siendo como posibles valores 0 o 1 (En activo o desactivado respectivamente). Condition es la variable condición para el Smart Contract, todo contrato trabaja sobre personas, una condición y un entregable, entonces, condition, es la variable que permite dar por finalizado o no el Smart Contract. DeliveredUniPoints es la variable que se comprueban con condition. En ella por cada transacción en la que tenga que actuar el Smart Contract, se actualiza este valor con los UniPoints que vaya recibiendo el usuario de la UniReward. Por último, el objeto a entregar tras cumplir el contrato es UniRewardId, este atributo almacena la Id de la UniReward que se entrega tras finalizar el contrato.

Como funciones de esta clase, se tienen:

```
setDeliveredUniPoints(deliveredUniPoints) {
    this.deliveredUniPoints = [...deliveredUniPoints];
}

calHashTransaction() {
    return crypto.createHash('sha256').update(this.walletIdDemander + this.walletIdObserver + this.state + this.condition + this.deliveredUniPoints +this.UniRewardId).digest('hex');
}

signContract(signingKey, type) {···
}

async terminateContract() {
    await controllerSContractDB.updateStateSC(this.UniRewardId)
    this.state = true
}

proveCompleteContract() {
    console.log('============================================================')
    console.log(this.deliveredUniPoints + "is equals to " + this.condition)
    if (this.deliveredUniPoints.length != this.condition.length) {
        return false
    } else {
        return (Array.isArray(this.deliveredUniPoints) &&
            Array.isArray(this.condition) &&
            this.deliveredUniPoints.length === this.condition.length &&
            this.deliveredUniPoints.every((val, index) => val === this.condition[index]))

    }
}

async endSmartContract(idsWallets, transactionObjId, idsToChange) {···
}
```

Al igual que en la clase transaction, se tienen dos funciones calHashContract y singContract. Esas funciones tienen la misma funcionalidad que calHashTransaction y singTransaction.

Como funciones nuevas en esta clase, están: setDeliveredUniPoints, se encarga de rellenar los UniPoints que han sido entregados ya al usuario. Así que, esta función rellena el atributo deliveredUniPoints con el contenido del parámetro de la función.

La función proveCompleteContract se encarga de comprobar que la condición y los UniPoints entregados al usuario sean los mismos. Se encarga de comprobar que deliveredUniPoints sea de tipo array, comprueba que ambos vectores tengan la misma longitud y que tengan los mismos valores. En función de estas comparaciones, se devuelve un valor a true o a false.

Las funciones terminateContract y endSmartContract, son funciones que se encargan de finalizar los contratos. La primera, se encarga de terminar los contratos en casos puntuales como: eliminación de un usuario y errores con el contrato. La segunda es la encarga de actualizar la UniReward, canjear los UniPoints y entregar la UniReward una vez se cumpla la condición del contrato, de dar por finalizado el contrato, entregar y firmar el contrato por ambos usuarios.

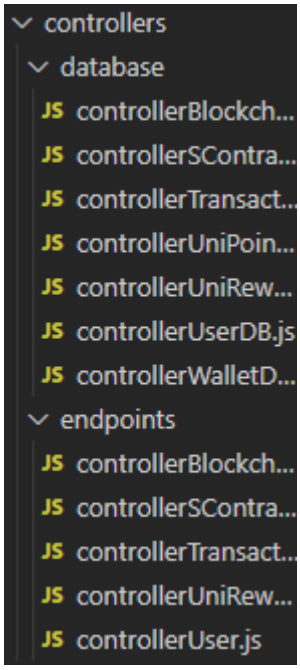
1.2 Controladores

En este proyecto se tienen dos tipos de controladores (controllers). Están los que almacenan las diferentes funcionalidades dentro de cada endpoint y los controladores que realizan las peticiones a base de datos.

Dichos controladores en el repositorio están localizados en la capeta controllers y divididos en database y endpoints.

En “controllers/database” se encuentran los controladores respectivos a todos los objetos/tablas de la base de datos: Blockchain, smart contracts, Ttansactions, UniPoints, UniRewards, user y wallets.

Por otra parte en “controllers/endpoints” se encuentran los métodos que aparecen en los diferentes endpoints. La única finalidad de los mismos es que la cantidad de código este distribuida en varias clases y no en una única clase. Las clases que se han elgido para este proyecto son controllerBlockchain, controllerSContract, controllerUniReward, controllerUser y controllerTransaction.



2. Endpoints

En este punto se procede a explicar los endpoints existentes en esta plataforma.para su comprensión y entendimiento:

2.1 Endpoints

Los primeros endpoints que se pueden observar en el fichero index.js de “routes”, son aquellos que pueden ser usados para el frontend para cosas funcionales como obtener una lista de datos o un dato en concreto

GetAllUserList	
Nombre de la ruta	/getAllUserList
Tipo de petición	GET
Parámetros	Ninguno
Salida concreta	Devuelve un array con los usuarios de la plataforma
Flujos alternativos	Si no hay usuarios en la plataforma, devuelve array vacío por defecto
Descripción	Con este endpoint, el usuario puede obtener una lista de todos los usuarios de la plataforma

GetUsersName	
Nombre de la ruta	/getUsersName/:id, donde id es el identificador numérico del usuario
Tipo de petición	GET
Parámetros	id→ number

Salida concreta	Devuelve un string con el nombre del usuario que tiene por identificador id
Flujos alternativos	Los errores son devueltos en forma de string Si no el id no es de tipo numérico: "User data don't loaded - Reason: No user to load data"
Descripción	Con este endpoint, se puede obtener el nombre de usuario en la plataforma de un usuario en concreto

GetUserRole	
Nombre de la ruta	/getUserRole/:id, donde id es el identificador numérico del usuario
Tipo de petición	GET
Parámetros	Id → number
Salida concreta	Devuelve un string con el tipo de rol del usuario que tiene por identificador id
Flujos alternativos	Los errores son devueltos en forma de string Si no el id no es de tipo numérico: "User data don't loaded - Reason: No user to load data"
Descripción	Con este endpoint, el usuario puede obtener el rol de un usuario de la plataforma en concreto

GetUserID	
Nombre de la ruta	/getUserID/:username, donde usermane es el identificador del nombre de usuario
Tipo de petición	GET
Parámetros	En un body JSON tienen que estar presentes los siguientes elementos 'username' → 'string'
Salida concreta	Devuelve un string con el id del usuario
Flujos alternativos	Los errores son devueltos en forma de string Si no existe alguno de los parámetros o es de tipo erroneo: “User username don't located - Reason: Not correct type of parameter”
Descripción	Con este endpoint, el usuario puede obtener el identificador de un usuario de la plataforma con un nombre en concreto

LoginUser	
Nombre de la ruta	/loginUser
Tipo de petición	POST
Parámetros	En un body JSON tienen que estar presentes los siguientes elementos 'username' → 'string' 'password' → 'string'
Salida concreta	Devuelve un string con el id del usuario
Flujos alternativos	Los errores son devueltos en forma de string

	Si no existe alguno de los parámetros o es de tipo erroneo: "Can't login - Reason: Not correct parameters" Si el usuario a loguear no existe: "Can't login - Reason: Don't exist a user with this password or username" Si el usuario esta eliminado: "Can't login - Reason: Don't exist a user with this password or username"
Descripción	Con este endpoint, el usuario puede “loguearse” en la plataforma (en el frontend) y poder realizar las diferentes opciones que brinda la plataforma

GetAllRewardsList	
Nombre de la ruta	/getAllRewardsList/:id/:purch, donde id es el identificador númerico del usuario y purch es un parámetro booleano para buscar entre las UniRewards por el campo purchase
Tipo de petición	GET
Parámetros	En un body JSON tienen que estar presentes los siguientes elementos id → ‘number’ purch → ‘boolean’
Salida concreta	Devuelve un string con el id del usuario
Flujos alternativos	Los errores son devueltos en forma de string Si no existe alguno de los parámetros o es de tipo erroneo: “User data don't loaded - Reason: No user to load data”

Descripción	Con este endpoint, se pueden obtener las diferentes UniRewards sin canjear enlazadas a un usuario en concreto

GetAllSmartContractsUser	
Nombre de la ruta	/getAllSmartContractsUser/:id
Tipo de petición	GET
Parámetros	Id → number
Salida concreta	Devuelve un string con el tipo de rol del usuario que tiene por identificador id
Flujos alternativos	Los errores son devueltos en forma de string Si no el id no es de tipo numérico: "User data don't loaded - Reason: No user to load data" Si no hay contratos para el usuario: "No courses are activated for the user"
Descripción	Con este endpoint, se pueden obtener todas los Smart Contrats en activo que un usuario en conreto posee

GetSpecificUser	
Nombre de la ruta	/getSpecificUser/:id
Tipo de petición	GET
Parámetros	Id → number
Salida concreta	Devuelve un archivo JSON con los datos personales del usuario
Flujos alternativos	Los errores son devueltos en forma de string

	Si no el id no es de tipo numérico: "User data don't loaded - Reason: No user to load data"
Descripción	Con este endpoint, se puede obtener toda la información referente a un usuario

GetSpecificWallet	
Nombre de la ruta	/getSpecificWallet/:id
Tipo de petición	GET
Parámetros	Id → number
Salida concreta	Devuelve un archivo JSON con los datos del monedero del usuario
Flujos alternativos	Los errores son devueltos en forma de string Si no el id no es de tipo numérico: "User data don't loaded - Reason: No user to load data"
Descripción	Con este endpoint, se puede obtener toda la información referente a un monedero de un usuario en concreto de la plataforma

Los demás endpoints que se observan en este fichero, son aquellos que que dan funcionalidad a la plataforma:

CreateNewReward	
Nombre de la ruta	/createNewReward
Tipo de petición	POST
Parámetros	En un body JSON tienen que estar presentes los siguientes elementos

	‘nameUR’ → 'string' ‘descriptionUR’ → 'string' ‘imageUR‘ → 'string' ‘costReward‘ → ‘number’ ‘username’→ 'string' 'password' → 'string' ‘usernameCourse’→ 'string'
Salida concreta	Devuelve un string con el siguiente contenido: “OK - Reward will be created"
Flujos alternativos	Los errores son devueltos en forma de string Si no existe alguno de los parámetros o es de tipo erróneo: "Can't login - Reason: Not correct parameters" Si la Blockchain no es correcta: " Reward not created - Reason: Blockchain isn't valid" Si se está ejecutando un reinicio de la base de datos: “Reward not created - Reason: Server is doing a restart” Si el usuario que está creando la UniReward no existe: Reward not created - Reason: Username or password isn't correct Si la cantidad de puntos es 0: "Reward not created - Reason: Can't create a free UniReward" o menor que 0: “Reward not created - Reason: System User for points transation dosen't exist” Si el usuario que está creando la UniReward está eliminado : “Reward not created - Reason: Username or password isn't correct"

	Si el usuario que está creando la UniReward no tiene el rol de instructor: "Reward not created - Reason: Username without permissions" Si el usuario intenta darse a sí mismo una UniReward: "Reward not created - Reason: Instructor can't give to itselft UniPoints" Si el usuario destinatario no existe: "Reward not created - Reason: User of course dosen't exist" El objeto UniReward no se creó: Reward not created - UniReward corrupted during the creation of object Fallan las firmas a la hora de realizar la transacción de los UniPoints al sistema: "Can't do the creation - Reason: Something go wrong during the sign of transaction"
Descripción	Con este endpoint, se puden crear las UniRewards en la plataforma, solo un usuario con rol instructor puede llevar a cabo esta operación

CreateNewTransaction	
Nombre de la ruta	/createNewTransaction
Tipo de petición	POST
Parámetros	En un body JSON tienen que estar presentes los siguientes elementos ‘fromAddressUN’ → 'string' ‘toAddressUN’ → 'string' ‘typeT‘ → 'string' ‘moneyTo’ → ‘number’ ‘concept’ → 'string' 'password' → 'string' ‘uniRewardId’ → 'string'
Salida concreta	Devuelve un string con el siguiente contenido: “OK - Delivery complete"
Flujos alternativos	Los errores son devueltos en forma de string Si no existe alguno de los parámetros o es de tipo erróneo: " Can't finish the Transaction - Reason: Not correct parameters" Si la Blockchain no es correcta: " Transaction not created - Reason: Blockchain isn't valid" Si se está ejecutando un reinicio de la base de datos: “Transaction not created - Reason: Server is doing a restart”

	Cuando el usuario destinatario o el emisor no existen: “Can't finish the Transaction - Reason: Destiny user or Emisor user dosen't exist” Si la cantidad de puntos es 0: " Can't do the payment - Reason: Amount of money can't be 0 cost or negative cost”. Si es negativo, el error que sale es el de parámetros no correctos: " Can't finish the Transaction - Reason: Not correct parameters". Si el instructor que está creando está eliminado: “Reward not created - Reason: Username or password isn't correct" Si el usuario que está creando la transacción no tiene el rol de instructor: "Reward not created - Reason: Username without permissions" Si el usuario que está creando la transacción intenta entregarse a si mismos puntos: "Can't finish the Transaction - Reason: User From and User Destiny can't be the same" Si el sistema intenta hacer una transacción sin la cantidad de UniPoints necesarios: "Can't do the payment - Reason: Amount of money in wallet is insuficient" Si el instructor intenta dar UniPoints a un usuario, referentes a una UniReward que no le pertenece: "Can't do the payment - Reason: Not the correct UniReward to delivery Unipoints to user" Si el instructor intenta dar UniPoints a un usuario, referentes a una UniReward que no le pertenece: "Can't do the payment - Reason: Not the correct UniReward to delivery Unipoints to user"

	Se intentan dar UniPoints referentes a una UniReward que no existe: "Can't do the payment - Reason: UniReward linked to reward dosen't exist" Fallan las firmas a la hora de realizar la transacción de los UniPoints al sistema: "Can't do the payment - Reason: Something go wrong during the sign of transaction"
Descripción	Con este endpoint, se pueden crear las diferentes transferencias de UniPoints por el desempeño en las sesiones de iVictrix. Tras completar tareas en concreto, un usuario puede entregar a otro los UniPoints referente a una UniReward en concreto. Solo un usuario de tipo instructor puede llevar a cabo esta operación y un instructor puede pasar UniPoints a otro instructor.

CreateNewUser	
Nombre de la ruta	/createNewUser
Tipo de petición	POST
Parámetros	En un body JSON tienen que estar presentes los siguientes elementos ‘name’ → 'string' ‘username’ → 'string' ‘fullSurname‘ → 'string' ‘password’ → ‘number’ ‘typeUser’ → 'string'
Salida concreta	Devuelve un string con el siguiente contenido: “OK – Account created"

Flujos alternativos	Los errores son devueltos en forma de string Si no existe alguno de los parámetros o es de tipo erróneo: "User dont created - Reason: The data of parameters isn't correct" Si se está ejecutando un reinicio de la base de datos: “User not created - Reason: Server is doing a restart” Si el usuario intenta poner un nombre de usuario ya usado en la plataforma: "Acount dont created - Reason: Username already is used" Si el usuario intenta poner un rol de usuario que no existe (Distinto de N o I): " User not created dont created - Reason: User role invalid"
Descripción	Con este endpoint, se pueden dar de alta a los nuevos usuarios de la plataforma.

ChangeUserData	
Nombre de la ruta	/changeUserData
Tipo de petición	POST
Parámetros	En un body JSON tienen que estar presentes los siguientes elementos ‘username’ → 'string' ‘password’ → 'string' ‘changes‘ → ‘object’, siendo un array de iniciales con los cambios concretos a realizar en el usuario, pudiendo ser como posibles valores de este array: [n,u,f,p] Y si hay cambios:

	nameN → 'string' usernameN → 'string' fullSurnameN → 'string' passwordN → 'string'
Salida concreta	Devuelve un string con el siguiente contenido: “OK – User data changed"
Flujos alternativos	Los errores son devueltos en forma de string Si no existe alguno de los parámetros o es de tipo erróneo: "User not modified - Reason: Parameters are not corrects" Si la contraseña y el nombre del usuario a cambiar no coinciden o el usuario está eliminado: “User data not changed - Reason: Server is doing a restart” Si existe algún valor de changes que no tenga su correpondiente key en los parametros de la petición: "User data dont change - Reason: Data to change are not correct" Si el nombre de usuario a cambiar coincide con alguno de la base de datos: “User data dont change - Reason: Already exists a user with this username”
Descripción	Con este endpoint, se pueden cambiar los datos personales de un usuario en concreto. Para ello, internamente habrá un array llamado changes como parametro de la petición. Este array guarda las iniciales de los cambios concretos que se van a hacer, si por ejemplo se cambian el nombre, los apellidos y la contraseña, este array tendrá el siguiente contenido: [n,f,p]. Junto al contenido de este array, se espera

	tambien que venga acompañado de sus respectivos parametros de cambio

DeleteUser	
Nombre de la ruta	/deleteUser
Tipo de petición	POST
Parámetros	En un body JSON tienen que estar presentes los siguientes elementos ‘id’ → 'number'
Salida concreta	Devuelve un string con el siguiente contenido: "OK - user‘s data eliminated", donde user es el nombre del usuario que elimina sus datos
Flujos alternativos	Los errores son devueltos en forma de string Si no existe alguno de los parámetros o es de tipo erróneo: "User can't be deleted - Reason: Not correct parammeters" Si el usuario ya esta eliminado: “user's data can't be eliminated - Reason: Already deleted”, donde user es el nombre del usuario que elimina sus datos Si el usuario a eliminar no existe: “user's data can't be eliminated - Reason: User Not Exist”, donde user es el nombre del usuario que elimina sus datos
Descripción	Con este endpoint, se puede “eliminar” toda la información referente a un usuario

A mayores de estos endpoints localizados en “routes/index.js”, se tiene otro fijero js que tambien contiene rutas alternativas. Estas rutas son usadas para el mantenimiento de la base de datos:

RestoreDB	
Nombre de la ruta	/maintenance/restoreDB
Tipo de petición	GET
Parámetros	No hay parametros
Salida concreta	Devuelve un objeto JSON cuyo contenido es: {ok : true}
Flujos alternativos	No tiene flujos alternativos más allá de que si surge algún error por no encontrar la base de datos o la tabla en concreto, finalizará la ejecución del servidor
Descripción	Con este endpoint, se puede hacer un reinicio de la base de datos, vaciando el contenido de la misma y volviendo a crear todas y cada una de las tablas

CreateUI	
Nombre de la ruta	/maintenance/createUI
Tipo de petición	GET
Parámetros	No hay parametros
Salida concreta	Devuelve un objeto JSON cuyo contenido es: {ok : true}
Flujos alternativos	No tiene flujos alternativos más allá de que si surge algún error por no encontrar la base de datos o la tabla en concreto, finalizará la ejecución del servidor
Descripción	Con este endpoint, se pueden crear un usuario normal y otro instrutor en la base datos. Se crean con nombres alumno e instructor respectivamente, para hacer pruebas si se desea con ellos

En cualquier caso en el que no se pueda conectar con la base de datos o no se haya realizado la configuración previa, se notifica al usuario que la

operación no puede realizarse por que la base de datos no esta bien. Se recomienda haber realizado los pasos previos en la Guía de instalacción entregada en el proyecto.

2.2 Funciones

A parte de los endpoints anteriormente detallados, cabe destacar el uso de varias funciones para el correcto funcionamiento de la plataforma. Entre estas funciones se encuentran:

Las funciones **addPendingTransaction** y **addPendingIds** se encargan de rellenar los arrays pendingTransaction y pendingIdsTransaction respectivamente. La primera función es la encargada de ir recogiendo todas las transacciones que se van realizando a lo largo del tiempo de ejecución del servidor, la segunda por su parte, se encarga de almacenar las ids de estas transacciones.

La función **proveKey**, se encuentra en el fichero "exports.js". Esta función se encarga de probar que un parametro en concreto este en el body de una petición y sea de un tipo esperado. Los parametros de esta función son, el nombre de la key que se quiere buscar y analizar dentro del body de la petición, el tipo de valor que se espera para ese parámetro y por último el body de la petición.

Por último, la función periodica o **periodicFunct**. Esta función se encarga de la creación de los objetos como: Transaction, UniRewards, UniPoints, Smart Contracts y los bloques de la Blockchain, son creados. Su nombre se debe a que es una función cíclica, se ejecuta concretamente cada 15 segundos en la plataforma. Esta función se desarrolla de la siguiente forma:

Primero se comprueba la cantidad de bloques que hay en la blockchain. Si no hay bloques en la Blockchain, se crea el bloque genesis o bloque origen, del que surgen los demás. Luego se crea un nuevo bloque con sus elementos correspondientes y con las ids de las transacciones que se han

creado durante esos 15 segundos, o el tiempo que crea oportuno el programador. Para poder cambiar este tiempo, se accede al archivo index.js de la carpeta routes, y en variable periodicFunc, se cambia la cantidad de segundos que se desean. Lo siguiente que hace la función es comprobar que haya UniRewards pendientes de creación y si las hay, se crean en base de datos junto a sus UniPoints correpondientes. A continuación, se cargan todas las transacciones pendientes de crear en base de datos. Posteriormente se actualizan los hashes de las transacciones, de las UniRewards y de los UniPoints con el valor del hash del bloque actual y cada id de la transacción correspondiente, quedando tal que así: hashBoqueActual+1, hashBoqueActual+2,…

Si hay transacciones que van enlazadas a la creación de UniRewards, en concreto, las del sistema dandose puntos a si mismo para poderlos entregar después a los usuarios, se crean sus Smart Contracts en base de datos, para poder tener un seguimiento de las UniRewards y su estado.

El siguiente paso, tras actualizar y crear toda la información, es comprobar el estado de los Smart Contracts que hay en la base de datos. Se hace una busqueda a base de datos, buscando todos los Smart Contracts que esten en activo y se recorren todos y cada uno de ellos buscando ver si condición y los UniPoints entregados son lo mismo, y si es así darlo por finalizado. Tambien existe la posibilidad de que si un usuario ha sido eliminado, terminar los contratos, a pesar de no cumplir los requisitos.