|  |
| --- |
|  |
| MANUAL TECNICO |
| [Subtítulo del documento] |

|  |
| --- |
|  |

Estudiantes

Cristina Jaqueline Herrera Rivas

Andrea Hernández Cervantes

Alberto Pablo Torres López

Cesar Daniel Cruz Correa

Rafael Sánchez Quirarte

Universidad Autónoma de la Laguna

Ingeniería en Sistemas Computacionales

Torreón, Coahuila

2022

CONTROL DE VERSIONES

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción | Autor |
| 28/04/2022 | 1.0 | Primera versión manual técnico de software | Cristina Jaqueline Herrera Rivas  Andrea Hernández Cervantes  Alberto Pablo Torres López  Cesar Daniel Cruz Correa  Rafael Sánchez Quirarte  Estudiantes de Ingeniería en Sistemas Computacionales  Universidad Autónoma de la Laguna |

DESCRIPCIÓN DEL DOCUMENTO

En el presente documento se muestra las técnicas que fueron aplicadas durante la ejecución del proyecto, con el fin de administrar un minado de criptomonedas mediante un bot.

Tener en cuenta que el fin de este manual es documentar las técnicas usadas durante el proyecto, y no como un curso de aprendizaje.

Contenido

[INTRODUCCION I](#_Toc101894367)

[OBJETIVO II](#_Toc101894368)

[1. ASPECTOS TECNICOS 1](#_Toc101894369)

[1.1 HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO 1](#_Toc101894370)

[1.1.1 GITHUB 1](#_Toc101894371)

[1.1.2 VISUAL STUDIO CODE 1](#_Toc101894372)

[1.1.3 MONGO DB 1](#_Toc101894373)

[1.1.4 NPM 2](#_Toc101894374)

[1.1.5 NODE JS 2](#_Toc101894375)

[1.1.6 NODEMON 3](#_Toc101894376)

[1.1.7 TAILWIND 3](#_Toc101894377)

[1.1.8 EXPRESS-GENERATOR 3](#_Toc101894378)

[1.1.9 EXPRESS 3](#_Toc101894379)

[1.1.10 EXPRESS-SESSION 4](#_Toc101894380)

[1.1.11 BCRYPT 4](#_Toc101894381)

[1.1.12 HBS 4](#_Toc101894382)

[1.1.13 PASSPORT 4](#_Toc101894383)

[1.1.14 PASSPORT-LOCAL 5](#_Toc101894384)

[1.1.15 CHART JS 5](#_Toc101894385)

[1.1.16 FETCH 5](#_Toc101894386)

[1.1.17 MONGUS 5](#_Toc101894387)

[1.1.18 JOI 5](#_Toc101894388)

[2. DIAGRAMAS DE MODELAMIENTO 6](#_Toc101894389)

[2.1 REGISTRAR UN NUEVO USUARIO 7](#_Toc101894390)

[2.1.1 PARAMETRO PARA LA SOLICITUD 7](#_Toc101894391)

[2.1.2 CODIGO 8](#_Toc101894392)

[2.1.3 DIAGRAMA DE FLUJO 10](#_Toc101894393)

[2.2 LOGIN DE ADMINISTRADOR Y USUARIO 11](#_Toc101894394)

[2.2.1 PARAMETRO PARA LA SOLICITUD 11](#_Toc101894395)

[2.2.2 CODIGO 12](#_Toc101894396)

[2.2.3 DIAGRAMA DE FLUJO 14](#_Toc101894397)

[2.2.4 14](#_Toc101894398)

[2.3 VER PAGINA DE INICIO POR PARTE DEL ADMINISTRADOR 15](#_Toc101894399)

[2.3.1 PARAMETRO PARA LA SOLICITUD 15](#_Toc101894400)

[2.3.2 CODIGO 16](#_Toc101894401)

[2.3.3 DIAGRAMA DE FLUJO 18](#_Toc101894402)

[2.4 VER PAGINA DE INICIO POR PARTE DEL USUARIO 19](#_Toc101894403)

[2.4.1 PARAMETRO PARA LA SOLICITUD 19](#_Toc101894404)

[2.4.2 CODIGO 20](#_Toc101894405)

[2.4.3 DIAGRAMA DE FLUJO 21](#_Toc101894406)

[3. REQUERIMIENTOS DEL SOFTWARE 22](#_Toc101894407)

[3.1 REQUISITOS MINIMOS 22](#_Toc101894408)

[1. REFERENCIAS 23](#_Toc101894409)

Contenido de tablas

[Tabla 1 Parámetros de solicitud para registro 7](#_Toc101894719)

[Tabla 2 Parámetro de solicitud para inicio de sesión 11](#_Toc101894720)

[Tabla 3 Parámetro de solicitud para vista principal de administrador 15](#_Toc101894721)

[Tabla 4 Parámetro de solicitud para vista principal de usuario 19](#_Toc101894722)

Contenido de diagramas

[Diagrama 1 Registro 10](file:///C:\Users\Andrea%20H\Documents\GitHub\DashboardCrypto\MANUAL%20TECNICO.docx#_Toc101894415)

[Diagrama 2 Inicio de sesión 14](file:///C:\Users\Andrea%20H\Documents\GitHub\DashboardCrypto\MANUAL%20TECNICO.docx#_Toc101894416)

[Diagrama 3 Vista principal administrador 18](file:///C:\Users\Andrea%20H\Documents\GitHub\DashboardCrypto\MANUAL%20TECNICO.docx#_Toc101894417)

[Diagrama 4 Vista principal usuario 21](#_Toc101894418)

# INTRODUCCION

El manual se realiza con el fin de detallar el “PROYECTO PANEL CRYPTO” en términos técnicos para que la persona que vaya a administrar, editar o configurar el software lo haga de una manera apropiada. El documento se encuentra dividido en las siguientes secciones.

Aspectos técnicos: Se darán a conocer conceptos, definiciones de los componentes del sistema desde el punto de vista teórico para mayor entendimiento por parte del lector sobre el funcionamiento del sistema de información y herramientas.

Aspecto técnico del desarrollo del sistema: Corresponde a la instrucción para el lector sobre los componentes del sistema desde una perspectiva técnica en los aspectos de almacenamiento de datos, estructura del desarrollo y recomendaciones del uso debido del sistema.

Requerimientos del software: Detalla los requerimientos básicos necesarios para el funcionamiento del software.

# OBJETIVO GENERAL

Re

# OBJETIVOS ESPECIFICOS

Mostrar al usuario

# ASPECTOS TECNICOS

El proyecto Panel Crypto tiene la finalidad de

## HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

### GITHUB

GitHub es una plataforma de alojamiento de código para colaboración y control de versiones.

Los esenciales de GitHub son:

Repositorios

* Branches
* Commits
* Pull Requests
* Git (el software de control de versiones en el que se basa GitHub)

### VISUAL STUDIO CODE

Visual Studio Code es un editor de código fuente que permite trabajar con diversos lenguajes de programación, admite gestionar tus propios atajos de teclado y refactorizar el código. Es gratuito, de código abierto y proporciona una utilidad para descargar y gestionar extensiones con las que se puede personalizar y potenciar esta herramienta.

Las extensiones de Visual Studio Code otorgan infinidad de opciones, como colorear tabulaciones, etiquetas o recomendaciones de autocompletado. También hay extensiones que ayudan con el lenguaje de programación que se vaya a utilizar, como por ejemplo para Python, C / C++, JavaScript.

### MONGO DB

MongoDB es un **sistema de base de datos NoSQL orientado a documentos de código abierto y escrito en C++**, que en lugar de guardar los datos en tablas lo hace en estructuras de datos BSON (similar a JSON) con un esquema dinámico. Al ser un proyecto de código abierto, sus binarios están disponibles para los sistemas operativos Windows, GNU/Linux, OS X y Solaris.

Esto representa una de las diferencias más importantes con respecto a las bases de datos relacionales. Y resulta que no es que no es necesario seguir un esquema. Los documentos de una misma colección - concepto similar a una tabla de una base de datos relacional -, pueden tener esquemas diferentes.

### NPM

Npm es la biblioteca de software más grande del mundo (Registro) npm también es un administrador e instalador de paquetes de software.

Todos los paquetes de npm se definen en archivos llamados package.json.El contenido de package.json debe estar escrito en JSON. Npm puede (en una línea de comando) instalar todas las dependencias de un proyecto y las dependencias también se definen en package.json .

### NODE JS

Node.js es un entorno controlado por eventos, diseñado para crear aplicaciones escalables, permitiéndote establecer y gestionar múltiples conexiones al mismo tiempo.

Las Apis de la biblioteca de Node.js son asíncronas, sin bloqueo. Un servidor basado en Node.js no espera que una API devuelva datos. El servidor pasa a la siguiente API después de llamarla, y un mecanismo de notificación de eventos ayuda al servidor a obtener una respuesta de la llamada a la API anterior.

Node.js utiliza un modelo de un solo subproceso con bucle de eventos. Gracias al mecanismo de eventos, el servidor responde sin bloqueos, esto hace que el servidor sea altamente escalable comparando con los servidores tradicionales como el Servidor HTTP de Apache.

### NODEMON

Nodemon es una herramienta que ayuda a desarrollar aplicaciones basadas en node.js al reiniciar automáticamente la aplicación del nodo cuando se detectan cambios en los archivos del directorio.

Nodemon no requiere ningún cambio adicional en su código o método de desarrollo. Nodemon es un contenedor de reemplazo para node. Para usar nodemon, reemplace la palabra node en la línea de comando cuando ejecute su script.

### TAILWIND

Según la documentación oficial de Tailwind CSS: “Tailwind CSS es un framework CSS de bajo nivel altamente personalizable que le permite crear diseños personalizados eliminando los estilos de componentes predefinidos que necesitarías sobrescribir en todo caso”.

Las interfaces de usuario personalizadas se pueden construir de manera efectiva usando CSS sin mucha codificación. Al ser un framework utility-first, Tailwind CSS ofrece la ventaja de diseñar cada componente de manera distintiva.

### EXPRESS-GENERATOR

Generador de aplicaciones de Express.

### EXPRESS

Express es el framework web más popular de Node, y es la librería subyacente para un gran número de otros frameworks web de Node populares. Proporciona mecanismos para:

* Escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes caminos URL (rutas).
* Integración con motores de renderización de "vistas" para generar respuestas mediante la introducción de datos en plantillas.
* Establecer ajustes de aplicaciones web como qué puerto usar para conectar, y la localización de las plantillas que se utilizan para renderizar la respuesta.
* Añadir procesamiento de peticiones "middleware" adicional en cualquier punto dentro de la tubería de manejo de la petición.

### EXPRESS-SESSION

El middleware express-session almacena los datos de sesión en el servidor; sólo guarda el ID de sesión en la propia cookie, no los datos de sesión. De forma predeterminada, utiliza el almacenamiento en memoria y no está diseñado para un entorno de producción.

### BCRYPT

El bcrypt es una implementación de JavaScript de la función de hash de contraseña bcrypt que le permite crear fácilmente un hash a partir de una contraseña string. A diferencia del cifrado que puede decodificar para recuperar la contraseña original, el hashing es una función unidireccional que no se puede revertir una vez realizada.

Cuando el usuario envía una contraseña, la contraseña se codificará y su aplicación de JavaScript debe almacenar el hash en la base de datos. Más tarde, cuando el usuario quiera autenticar su cuenta, debe comparar la contraseña ingresada con el hash almacenado en su base de datos para ver si coincide.

La librería bcrypt facilita el proceso al proporcionarle métodos para codificar y comparar contraseñas.

### HBS

Handlebars.js es un motor de plantillas. Se basa en el lenguaje de plantillas Moustache, Handlebars, separa el diseño HTML del resto de JavaScript, para así escribir un código mucho más limpio.

### PASSPORT

Passport es un middleware de autenticación compatible con Express para Node.js. El único propósito de Passport es autenticar las solicitudes, lo que hace a través de un conjunto extensible de complementos conocidos como estrategias. Passport no monta rutas ni asume ningún esquema de base de datos en particular, lo que maximiza la flexibilidad y permite que el desarrollador tome decisiones a nivel de aplicación. La API es simple: le proporciona a Passport una solicitud de autenticación y Passport proporciona ganchos para controlar lo que ocurre cuando la autenticación tiene éxito o falla.

### PASSPORT-LOCAL

Estrategia de passport para autenticarse con un nombre de usuario y contraseña. Este módulo le permite autenticarse usando un nombre de usuario y contraseña en sus aplicaciones Node.js. Al conectarse a Passport, la autenticación local se puede integrar de manera fácil y discreta en cualquier aplicación o marco que admita el middleware de estilo Connect, incluido Express.

### CHART JS

Chart.js es una librería JavaScript que permite generar distintos tipos de gráficas. Es posible representar datos usando 6 tipos de gráficas diferentes, totalmente personalizables y animadas.

### FETCH

La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, tales como peticiones y respuestas. También proporciona un método global fetch () que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red.

### MONGUS

### JOI

# DIAGRAMAS DE MODELAMIENTO

El direccionamiento hace referencia a la determinación de cómo responde una aplicación a una solicitud de cliente en un determinado punto final, que es un URI (o una vía de acceso) y un método de solicitud HTTP específico (GET, POST, etc.).

Cada ruta puede tener una o varias funciones de manejador, que se excluyen cuando se correlaciona la ruta.

La definición de ruta tiene la siguiente estructura:

app. METHOD (PATH, HANDLER)

Donde:

app es una instancia de express.

METHOD es un método de solicitud HTTP.

PATH es una vía de acceso en el servidor.

HANDLER es la función que se ejecuta cuando se correlaciona la ruta.

## REGISTRAR UN NUEVO USUARIO

Crear un nuevo usuario por parte del Administrador.

Método: POST

### PARAMETRO PARA LA SOLICITUD

|  |  |
| --- | --- |
| **nombre**  Localización: body  Nombre del usuario | Requerido  String |
| **apellido**  Localización: body  Apellido del usuario | Requerido  String |
| **email**  Localización: body  Email del usuario | Requerido  String |
| **password**  Localización: body  Contraseña del usuario | Requerido  String |

Tabla 1 Parámetros de solicitud para registro

### CODIGO

var express = require('express');

const createHttpError = require('http-errors');

var router = express.Router();

var { client, dbName } = require('../db/mongo');

const User = require('../models/user.model');

const { authSchema } = require('../validations/user.validation');

const bcrypt = require('bcryptjs');

/\* GET users listing. \*/

router.get('/', function (req, res, next) {

res.render('signupAdmin');

});

router.post("/registro", async (req, res, next) => {

const result = await authSchema.validateAsync(req.body)

regUser(result)

.then(async () => {

res.render('usersAdmin')

})

.catch((err) => {

console.log(err);

})

.finally(() => {

client.close;

})

});

async function regUser(datos) {

await client.connect();

const db = client.db(dbName);

const collection = db.collection('Usuarios');

const hashUserPassword = await bcrypt.hash(datos.password, 10)

const users = new User({

nombre: datos.nombre,

apellido: datos.apellido,

email: datos.email,

password: hashUserPassword

}

)

await collection.insertOne(users)

module.exports = router;

En la ruta de registro se tiene dos métodos la primera ruta es con el método GET para renderizar la vista, donde se hará el registro de usuario.

La segunda ruta con el método POST es para obtener los datos de registro, mandados desde la función async *regUser,* donde se hace la conexión a la base de datos y a su colección *Usuarios*.

Con el método *insertOne* se insertan los datos que están dentro del objeto {nombre, apellido, email, password}, la contraseña esta encriptada mediante la librería *bcrypy* de npm.

Dentro del método POST se manda a llamar la constante *authSchema*, que hace referencia a la validación con *joi*, donde se valida que los datos del objeto {nombre, apellido, email, password} tienen los requerimientos del modelo *Schema* por parte de *mongoose.*

### DIAGRAMA DE FLUJO

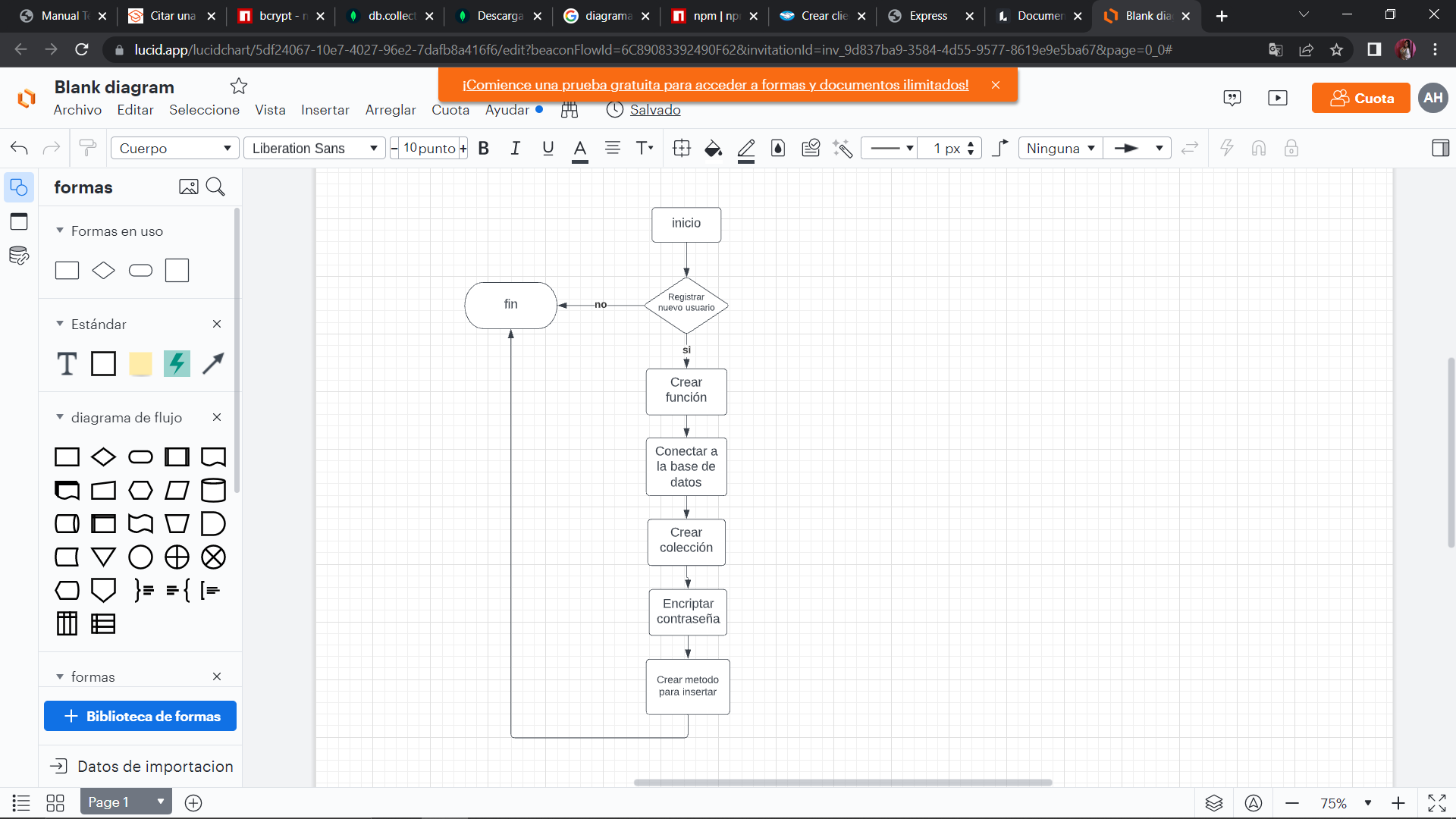


Diagrama 1 Registro

## LOGIN DE ADMINISTRADOR Y USUARIO

Iniciar sesión dependiendo si las credenciales son con permisos de usuario o administrador.

Método: POST

### PARAMETRO PARA LA SOLICITUD

|  |  |
| --- | --- |
| **username**  Localización: body  Correo del usuario | Requerido  String |
| **password**  Localización: body  Contraseña del usuario | Requerido  String |

Tabla 2 Parámetro de solicitud para inicio de sesión

### CODIGO

var express = require('express');

var router = express.Router();

var { client, dbName } = require("../db/mongo");

var passport = require('passport');

var LocalStrategy = require('passport-local');

const bcrypt = require('bcryptjs');

passport.use(new LocalStrategy(

async function (username, password, done) {

await client.connect();

const db = client.db(dbName);

const collection = db.collection("Usuarios");

await collection.findOne({ email: username }, async function (err, user) {

if (err) { return done(err); }

if (!user) { return done(null, false); }

const isValidPass = bcrypt.compareSync(password, user.password);

if (!isValidPass) { return done(null, false); }

return done(null, user);

});

}

));

passport.serializeUser(function (user, done) {

done(null, user.\_id);

});

/\* GET users listing. \*/

router.get('/', function (req, res, next) {

res.render('Login', { title: 'Login' });

});

router.post('/login',

passport.authenticate('local', { failureRedirect: '/' }),

function (req, res) {

if (req.user.Profile === 'admin') {

res.redirect('/admin/Inicio');

}

else {

res.redirect('/usuario/Inicio');

}

});

module.exports = router;

Para la ruta del login se requieren las variables de express, del cliente, la base de datos, passport y passport-local, y para encriptar la contraseña, la librería bcrypt.

Con Passport tenemos la sesión y validamos con Passport-local , dentro de esta función asíncrona se conecta a la base de datos y dentro de la colección *Usuarios* se hace la búsqueda de parámetro *username* mediante el método de *findOne(),* en la misma función se valida la contraseña y el hash que se creó en la encriptación

Se renderiza la pagina del login por el método GET.

Mediante el método POST autenticamos la sesión donde se hace una comprobación con una función IF donde si el perfil es admin, el usuario se mantiene en la ruta /admin/Inicio, por el contrario, se redirigirá a la ruta /usuario/Inicio

.

### DIAGRAMA DE FLUJO

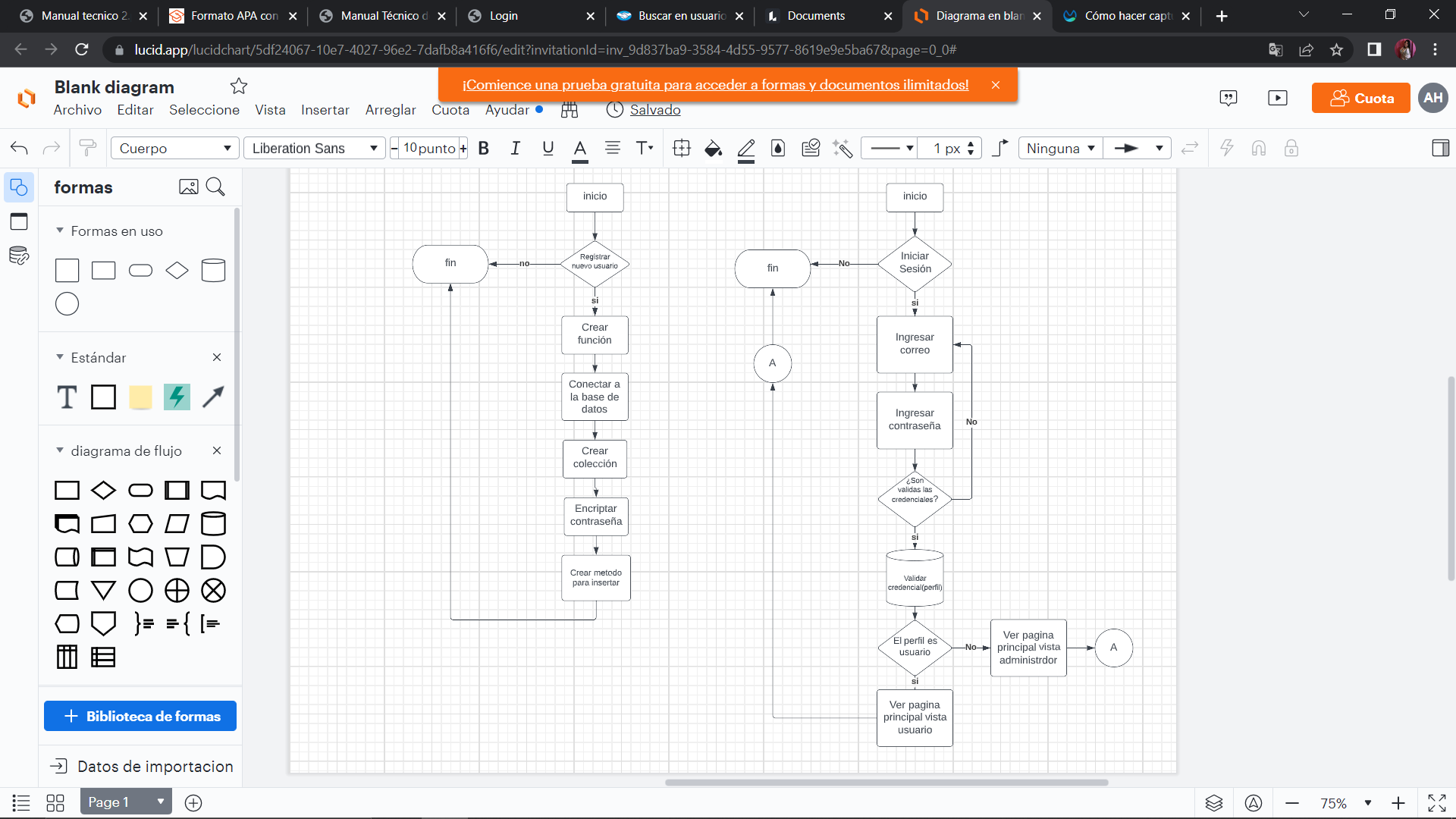


Diagrama 2 Inicio de sesión

## VER PAGINA DE INICIO POR PARTE DEL ADMINISTRADOR

Al iniciar sesión y validando las credenciales como administrador, entrará a la página principal modo administrador.

Método: GET

### PARAMETRO PARA LA SOLICITUD

|  |  |
| --- | --- |
| **profile**  Localización: base de datos  Perfil del usuario siendo administrador | Requerido  String |

Tabla 3 Parámetro de solicitud para vista principal de administrador

### CODIGO

var express = require('express');

var router = express.Router();

var passport = require('passport');

var {client, dbName} = require("../db/mongo");

var {ObjectId} = require('mongodb')

passport.deserializeUser(

async function(id, done) {

await client.connect();

const db = client.db(dbName);

const collection = db.collection("Usuarios");

collection.findOne({\_id: ObjectId(id)}, function (err, user) {

console.log(user)

done(err, user);

});

});

/\* GET home page. \*/

router.get('/',function(req, res, next){

if (req.isAuthenticated()) {

return next();

} else {

res.redirect('/')

}

}, function(req, res, next) {

if(req.user.Profile==="admin"){

res.render('indexAdmin', { title: 'Express' });

}

else{

res.redirect('/usuario/inicio')

}

});

module.exports = router;ç

El código aplica la estrategia de passport para autenticar las credenciales del usuario, en este caso, validar que realmente las credenciales introducidas pertenecen al administrador.

Con el método GET se redirige a la misma vista si es que las credenciales están validadas como administrador, haciendo una sentencia IF con la condición en la que, si el perfil del usuario es administrador, entonces renderizar la vista principal de administrador y si no se redirigirá a la ruta que le corresponde en este caso, a la principal de usuario.

### DIAGRAMA DE FLUJO

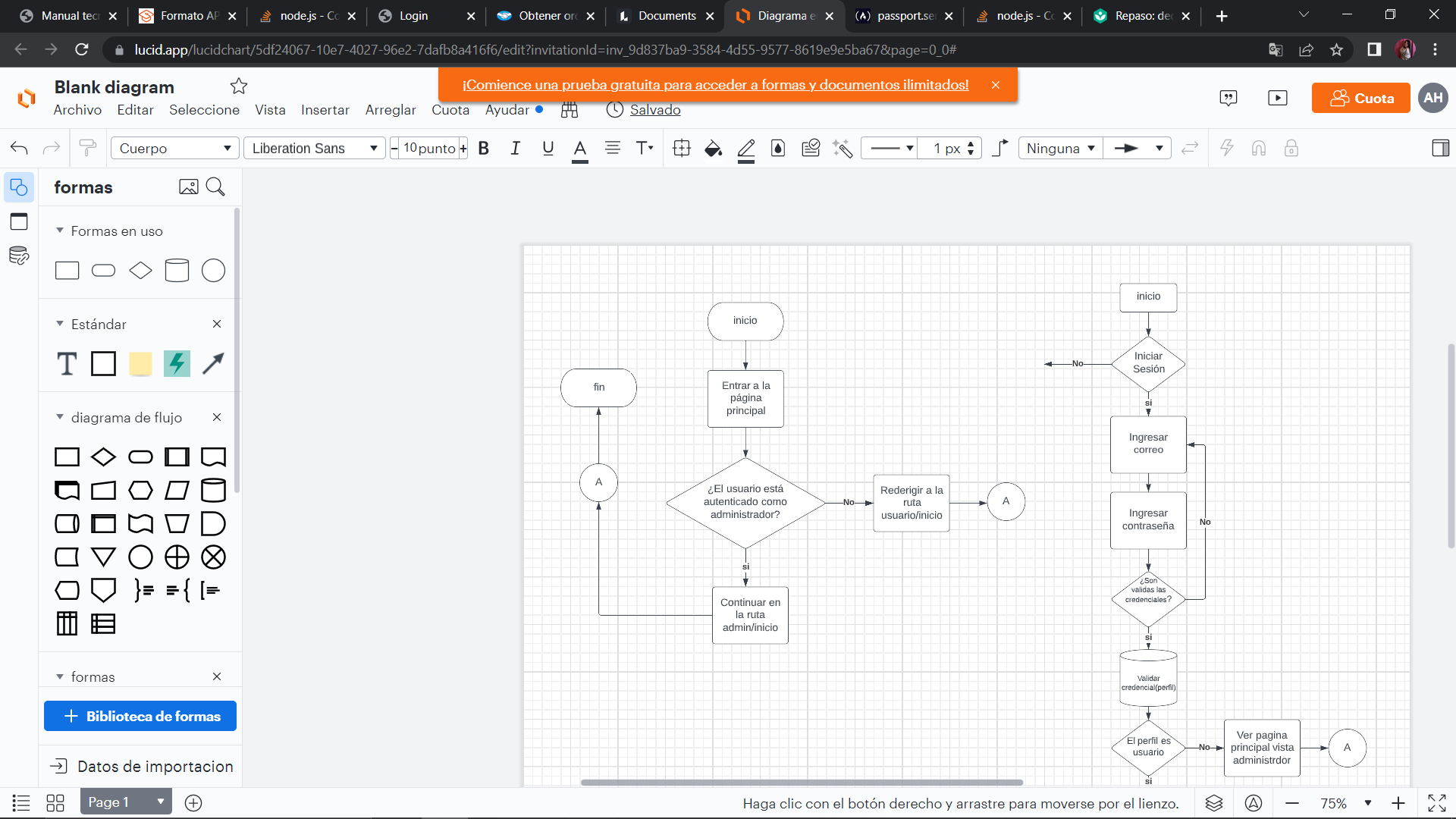


Diagrama 3 Vista principal administrador

## VER PAGINA DE INICIO POR PARTE DEL USUARIO

Al iniciar sesión y validando las credenciales como usuario, entrará a la página principal modo usuario.

Método: GET

### PARAMETRO PARA LA SOLICITUD

|  |  |
| --- | --- |
| **profile**  Localización: base de datos  Perfil del usuario siendo usuario | Requerido  String |

Tabla 4 Parámetro de solicitud para vista principal de usuario

### CODIGO

### DIAGRAMA DE FLUJO

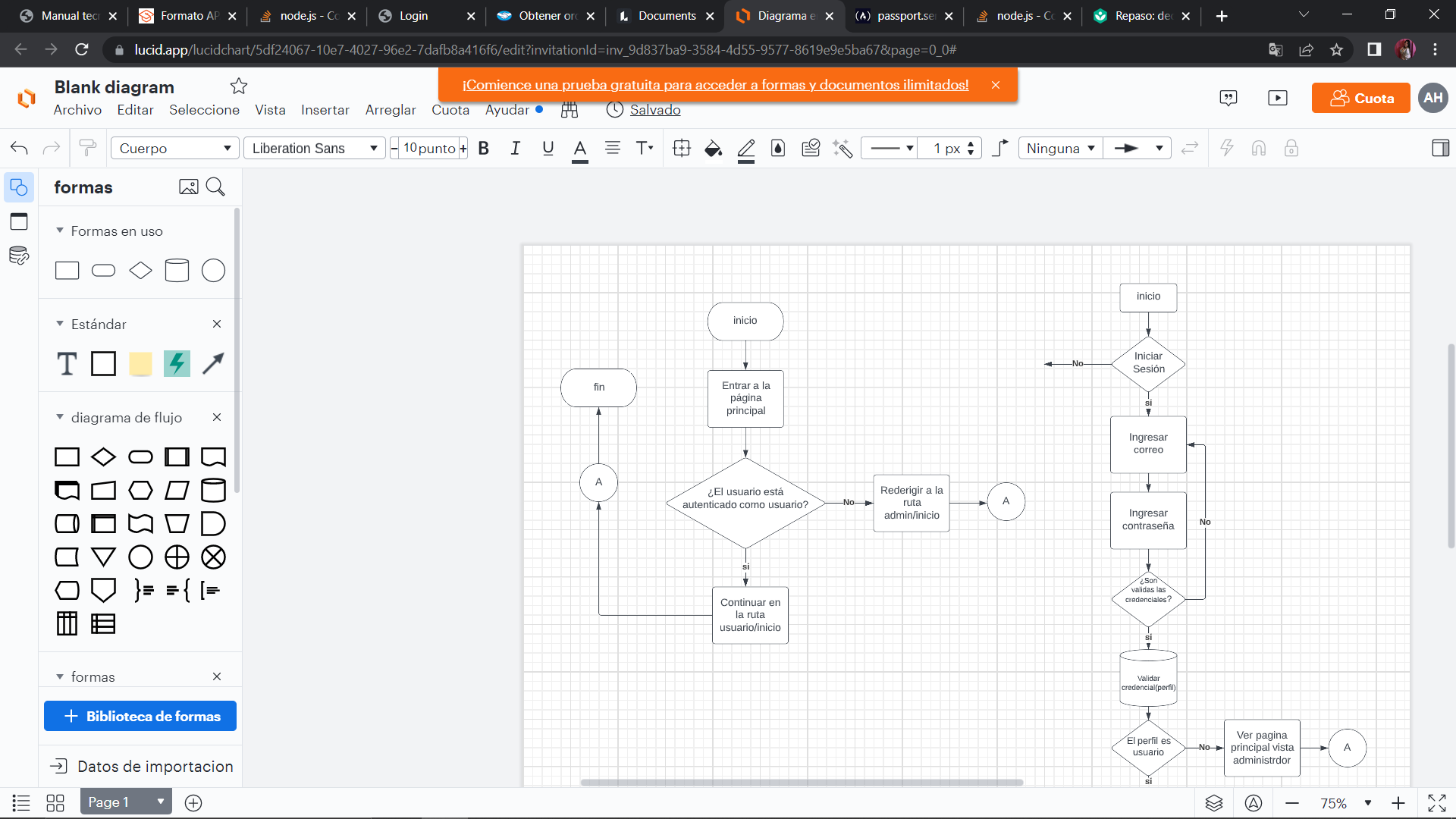


Diagrama 4 Vista principal usuario

## VER PÁGINA DE FINANZAS POR PARTE DEL ADMINISTRADOR

Ver el contendido de las finanzas de cada usuario.

Método: GET

### PARAMETRO PARA LA SOLICITUD

En esta solicitud no se requiere parámetros para la solicitud

### CODIGO

var express = require('express');

var router = express.Router();

/\* GET users listing. \*/

router.get('/', function(req, res, next) {

res.render('finanzasAdmin', { title: 'Express' });

});

module.exports = router;

### DIAGRAMA DE FLUJO

## VER PÁGINA DE FINANZAS POR PARTE DEL USUARIO

### PARAMETRO PARA LA SOLICITUD

### CODIGO

### DIAGRAMA DE FLUJO

## VER PÁGINA DE RIGS POR PARTE DEL ADMINISTRADOR

### PARAMETRO PARA LA SOLICITUD

### CODIGO

### DIAGRAMA DE FLUJO

## VER PÁGINA DE RIGS POR PARTE DEL USUARIO

### PARAMETRO PARA LA SOLICITUD

### CODIGO

### DIAGRAMA DE FLUJO

## VER PÁGINA DE USUARIOS POR PARTE DEL ADMINISTRADOR

### PARAMETRO PARA LA SOLICITUD

### CODIGO

### DIAGRAMA DE FLUJO

# REQUERIMIENTOS DEL SOFTWARE

Sistema Operativo: Windows 7

Procesador:

Memoria RAM:

Disco Duro:

Resolución de Pantalla:

Periféricos:

# Referencias

*GitHub* . (s.f.). Recuperado el 26 de Abril de 2022, de https://github.com/expressjs/generator

*Tailwind* . (s.f.). Recuperado el 25 de Abril de 2022, de https://www.atsistemas.com/es/blog/que-es-tailwind

(2021 de Noviembre de 2021). Obtenido de npm: https://www.npmjs.com/package/nodemon

*GitHub*. (s.f.). Recuperado el 25 de Abril de 2022, de W3 Schools: https://www.w3schools.com/whatis/whatis\_github.asp

J. (s. f.-b). GitHub - jaredhanson/passport-local: Username and password authentication strategy for Passport and Node.js. GitHub. Recuperado 26 de abril de 2022, de <https://github.com/jaredhanson/passport-local>

expressjs. (s. f.-b). GitHub - expressjs/session: Simple session middleware for Express. GitHub. Recuperado 26 de abril de 2022, de <https://github.com/expressjs/session>

expressjs. (s. f.-a). GitHub - expressjs/express: Fast, unopinionated, minimalist web framework for node. GitHub. Recuperado 26 de abril de 2022, de <https://github.com/expressjs/express>

Uso de Fetch - Referencia de la API Web | MDN. (2021, 11 febrero). Developer Mozilla. Recuperado 26 de abril de 2022, de <https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Using_Fetch>