

UNIVERSIDAD TECNOLÓGICA METROPOLITANA DE AGUASCALIENTES

Integrantes del equipo:

José De Jesús Sanchez Hernández

Nathalie Michelle Rodríguez Mireles

Jéssica Danaé Yáñez Esparza



Docente:

José De Jesús Salazar Padilla

Asignatura:

Desarrollo Móvil Integral

Grado y Grupo:

11 ° “A”

Nombre del documento:

Recopilación de entregables

Nombre de la aplicación:

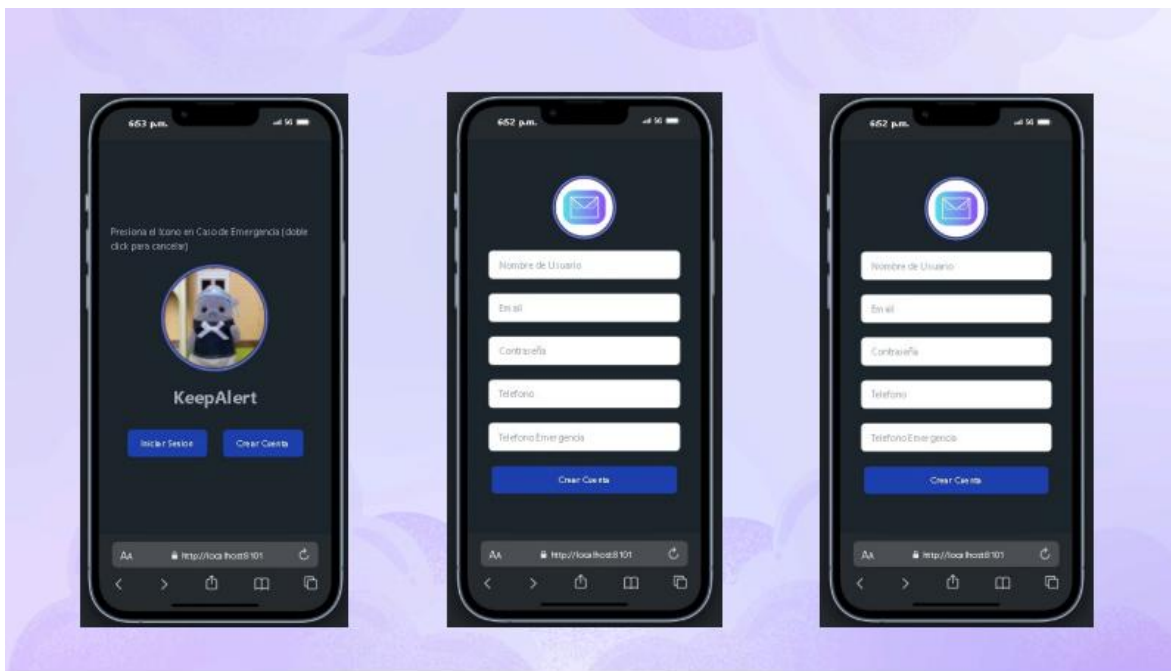
WONKY

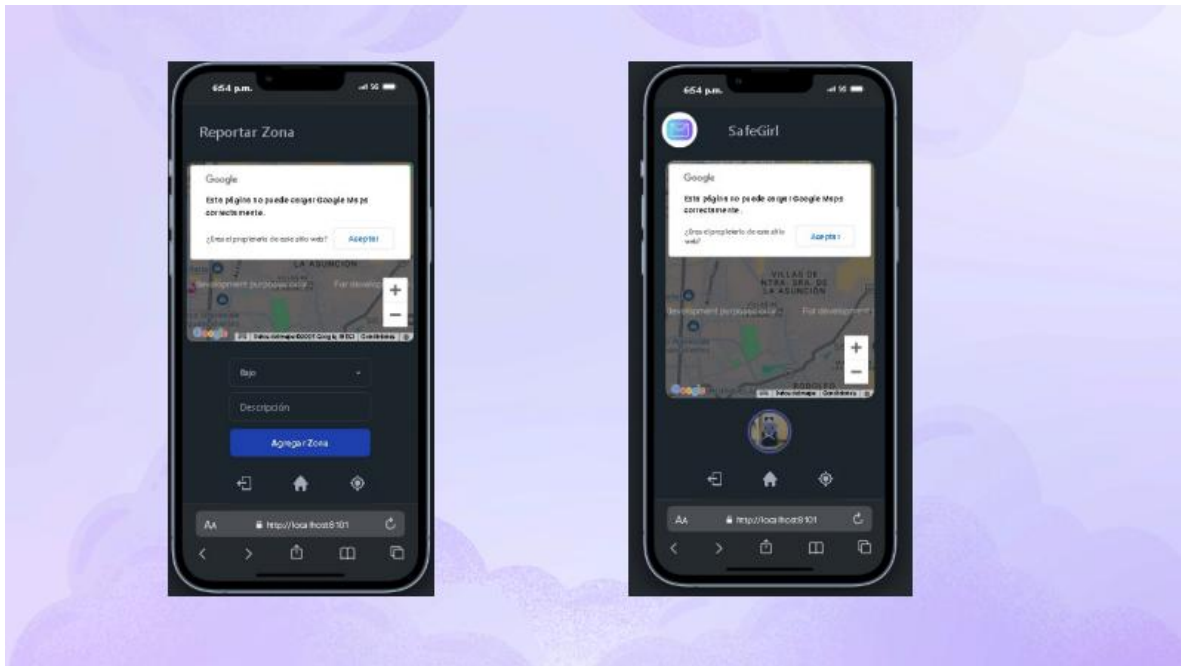
Fecha de entrega:

09/02/2025

Entregable #1

Esta es el primer paso en el cual creamos los mockups de nuestra aplicación, para tener nuestro punto de partida con el fin de tener en cuenta los posibles cambios y mejoras.









Herramientas utilizadas: FIGMA y CANVA

Entregable #2

Funciones UI

Pantalla	Descripción
	<p>En esta pantalla se muestra la pantalla de inicio en donde el usuario puede ingresar a nuestra aplicación.</p> <p>Tenemos dos botones uno para que el usuario inicie sesión y otro para crear una cuenta, los cuales nos redireccionan a otra ventana.</p> <p>También tenemos una etiqueta en donde nos da la instrucción de que el logo es un botón de pánico y nos indica en caso de cancelar solo dar doble clic.</p>
	<p>Esta pantalla es en donde vas a poder iniciar sesión, o poder registrarte en caso de no tener una cuenta</p> <p>Si se tiene una cuenta sólo tendrás que ingresar tu correo y tu contraseña, posteriormente darás clic en "iniciar sesión".</p> <p>Si no se cuenta con una cuenta darás clic en "Crear una cuenta" para registrar tus datos.</p>

	<p>Esta captura es el formulario que deberás responder para tener una cuenta en la aplicación en el cual pide lo siguiente:</p> <ul style="list-style-type: none"> *Nombre de usuario *E-mail *Contraseña *Teléfono *Teléfono de emergencia <p>Y posteriormente dar clic en "Crear cuenta", así podrás usar nuestra aplicación y cada vez que entres sólo deberás registrar tu correo y tu contraseña.</p>
	<p>Esta captura nos muestra cómo los usuarios van a realizar un reporte. Contiene un botón de agregar zona, nos da el nivel de riesgo y la descripción del tipo de peligro ya sea robo, violencia, etc.</p> <p>y nos muestra los botones para volver a salir.</p>
	<p>Esta es la interfaz principal en la cual nos muestra el mapa en el cual aparecerán los tipos de riesgo para que el usuario sepa cuáles son las zonas más seguras para transitar.</p>

Funciones Lógicas

- **Login:** El sistema al hacer el login envía una solicitud http al servidor backend, que primero va a validar las entradas y lo va a comparar con la información de la base de datos, una vez se encuentra que el usuario existe,

el sistema envía un token seguro para permitir el inicio de sesión.

- Registro: En el apartado de registro se envía una solicitud http que primero valida que las entradas sean correctas y no se agreguen caracteres invalidos o información vacia al servidor. Una vez hecho esto el servidor backend manda a guardar los datos capturados a la base de datos.
- Botón de emergencia: El botón de emergencia manda una alerta antes de redirigir al usuario a whatsapp, la limitante de este mensaje es que no se puede hacer 100% automático ya que whatsapp limita esta actividad.
- Reporte de la zona: Al entrar al apartado de reportes y que el usuario seleccione la ubicación a reportar o mande su ubicación en tiempo real, se manda la solicitud validada al backend, que posteriormente manda a guardar a la base de datos el reporte con su información.
- Carga de reportes: De manera asíncrona el sistema manda una solicitud para obtener toda la información de los reportes, esto para poder mostrarlos en el mapa.

.....

Entregable #3

Creación del cronograma:

[illegible]

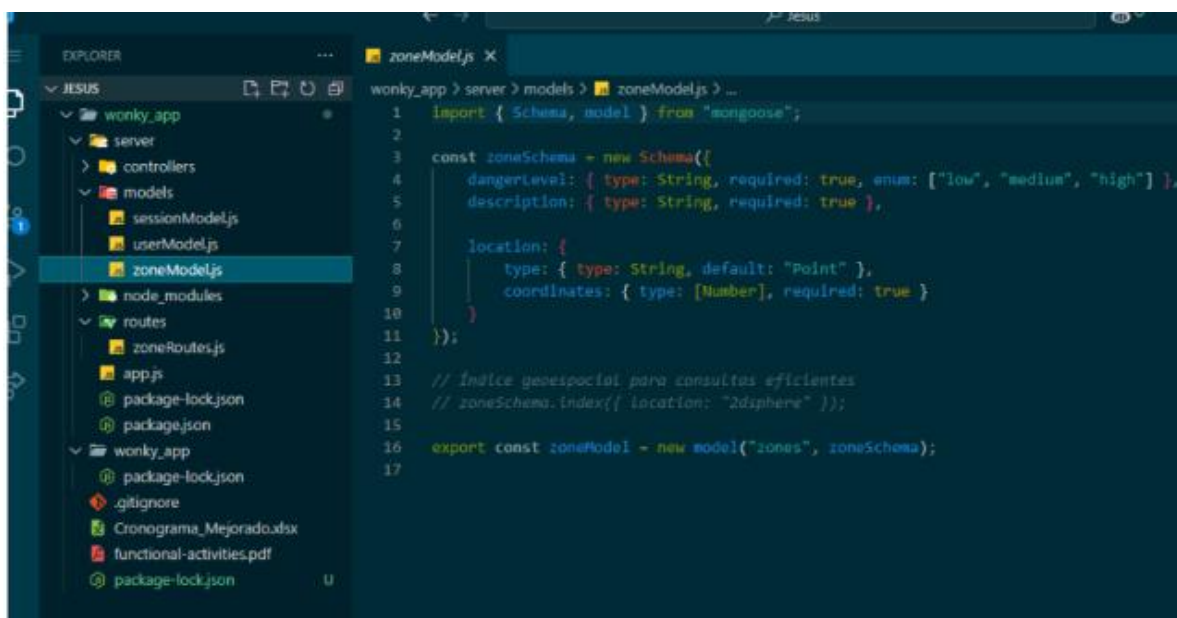
Entregable #4

```
1 import { Schema, model } from "mongoose";
2
3 const sessionSchema = new Schema({
4   userId: { type: Schema.Types.ObjectId, ref: "User", required: true },
5   jwt: { type: String, required: true },
6   createdAt: { type: Date, default: Date.now, expires: "1d" }
7 });
8
9 export const sessionModel = new model("sessions", sessionSchema);
10
```

```

1 import { Schema, model } from "mongoose";
2
3 const userSchema = new Schema({
4   name: { type: String, required: true },
5   apePat: { type: String, required: true },
6   apeMat: { type: String, required: true },
7   email: { type: String, required: true, unique: true },
8   password: { type: String, required: true },
9   numberPhone: { type: String, required: true, unique: true },
10
11   address: {
12     streetName: { type: String, required: true },
13     subdivision: { type: String, required: true },
14     number: { type: Number, required: true }
15   },
16
17   emerContact: {
18     name: { type: String, required: true },
19     email: { type: String, required: true },
20     numberPhone: { type: String, required: true }
21   }
22 });
23
24 export const userModel = mongoose.model("users", userSchema);

```



```

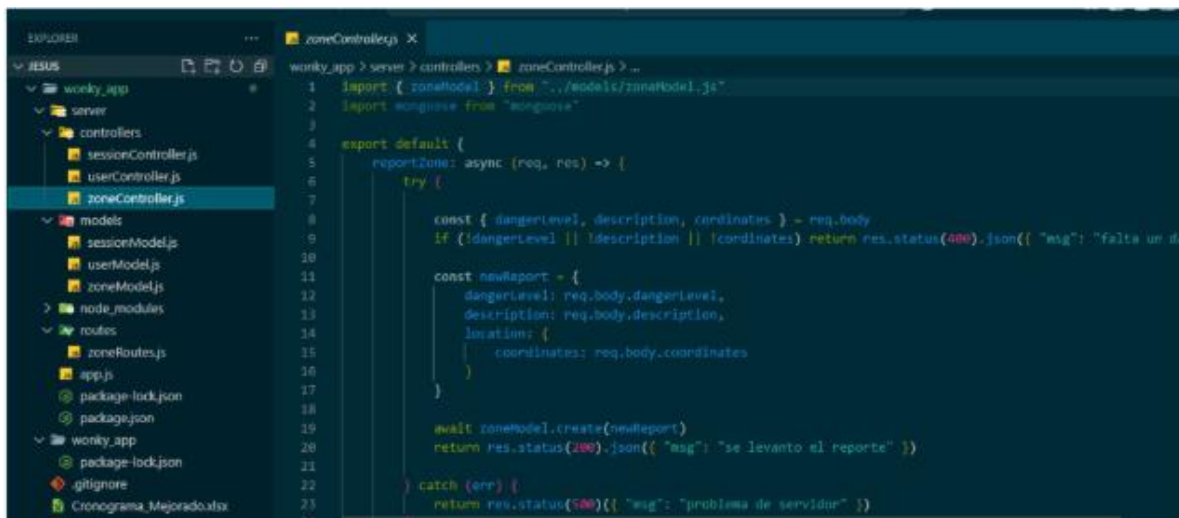
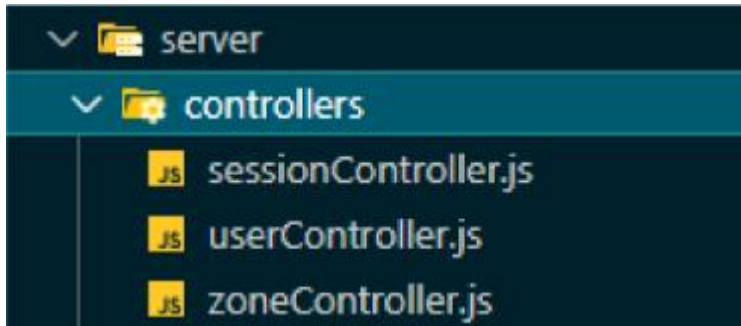
1 import { Schema, model } from "mongoose";
2
3 const zoneSchema = new Schema({
4   dangerLevel: { type: String, required: true, enum: ["low", "medium", "high"] },
5   description: { type: String, required: true },
6
7   location: {
8     type: { type: String, default: "Point" },
9     coordinates: { type: [Number], required: true }
10  }
11 });
12
13 // Índice geoespacial para consultas eficientes
14 // zoneSchema.index({ location: "2dsphere" });
15
16 export const zoneModel = new model("zones", zoneSchema);
17

```

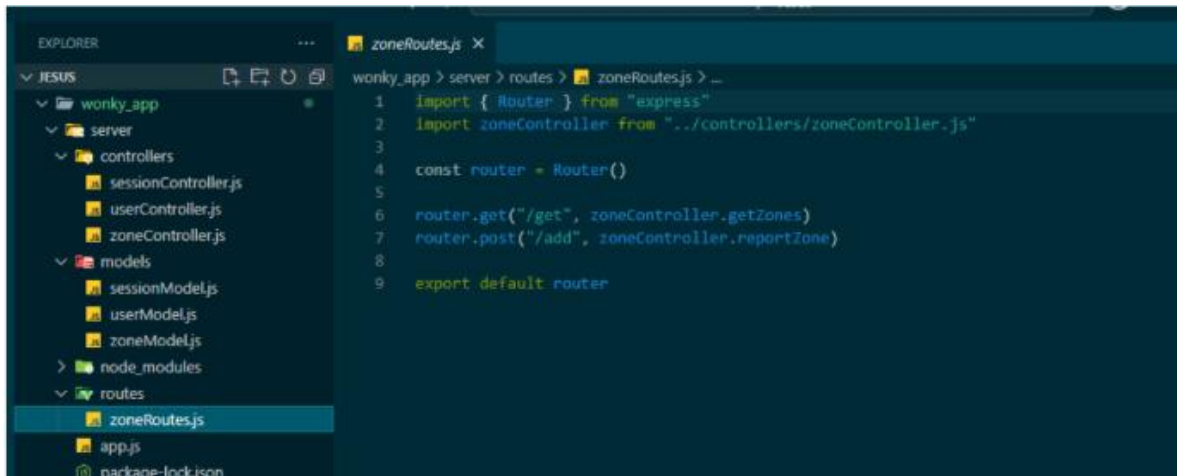
Se han incorporado nuevos modelos dentro del sistema, diseñados específicamente para mejorar la experiencia del usuario. Estos nuevos modelos incluyen características adaptadas tanto para los usuarios como para la gestión de las zonas, así como para optimizar el proceso de inicio de sesión. La implementación de estos modelos permitirá una mayor personalización y funcionalidad en cada una de estas áreas, asegurando que los usuarios puedan acceder de manera más eficiente y

fluida a sus cuentas, al mismo tiempo que se facilita la administración y asignación de zonas. Este cambio busca mejorar la seguridad, la facilidad de uso y la integración de diferentes componentes del sistema, garantizando una interacción más dinámica y precisa.

.....



Se ha hecho una especificación detallada del controlador correspondiente a la zona, con el objetivo de mejorar la asignación y administración de las mismas. Esta actualización facilita la interacción con los controladores al ofrecer una estructura más precisa y personalizada, lo que, a su vez, optimiza el rendimiento general del sistema y asegura que las zonas sean gestionadas de forma más eficiente y adecuada a las necesidades del usuario.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'server', 'controllers', 'models', 'routes', and 'app.js'. The 'routes' folder is expanded, showing 'zoneRoutes.js'. The code editor shows the content of 'zoneRoutes.js' with the following code:

```
1 import { Router } from "express"
2 import zoneController from "../controllers/zoneController.js"
3
4 const router = Router()
5
6 router.get("/get", zoneController.getZones)
7 router.post("/add", zoneController.reportZone)
8
9 export default router
```

Asimismo, se ha incorporado un nuevo apartado dedicado a la ruta, con el propósito de mejorar la interacción y la funcionalidad del sistema. Esta nueva sección permitirá a los usuarios obtener y enviar información de manera más eficiente al utilizar el mapa, facilitando la visualización de los riesgos y las notificaciones en tiempo real. Gracias a esta actualización, los usuarios podrán acceder a datos relevantes de forma más ágil, lo que contribuirá a una mejor toma de decisiones y a una experiencia de uso más fluida al interactuar con el mapa y los eventos asociados a él.

.....

Entregable #5

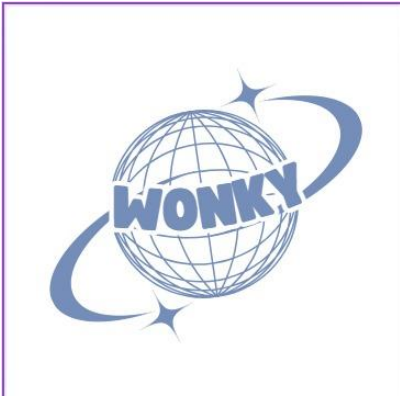
Logos





(Nota: Este logo se creó con IA por eso la falta de ortografía)





```

JESUS
wonky_app
├── server
│   ├── controllers
│   │   ├── sessionController.js
│   │   ├── userController.js
│   │   └── zoneController.js
│   ├── models
│   │   ├── sessionModel.js
│   │   ├── userModel.js
│   │   └── zoneModel.js
│   ├── node_modules
│   ├── routes
│   │   ├── userRoutes.js
│   │   ├── zoneRoutes.js
│   │   └── app.js
│   ├── package-lock.json
│   └── package.json
└── wonky_app
    ├── package-lock.json
    ├── .gitignore
    ├── Cronograma_Mejorado.xlsx
    ├── functional-activities.pdf
    └── package-lock.json

wonky_app > server > controllers > userController.js > ...
1  import { userModel } from "../models/userModel.js"
2  import { sessionModel } from "../models/sessionModel.js"
3  import bcrypt from "bcrypt"
4  import jwt from "jsonwebtoken"
5  import dotenv from "dotenv"
6
7  dotenv.config()
8
9  export default {
10     registerUser: async (req, res) => {
11         try {
12
13             const { name, apePat, apeMat, email, password, numberPhone } = req.body
14             if (!name || !apePat || !apeMat || !email || !password || !numberPhone) return res.status(400).json({ msg: "Faltan datos" })
15             const { streetName, subdivision, number } = req.body.address
16             if (!streetName || !subdivision || !number) return res.status(400).json({ msg: "algo mal con la direccion" })
17
18             const newUser = {
19                 name: name,
20                 apePat: apePat,
21                 apeMat: apeMat,
22                 email: email,
23                 numberPhone: numberPhone,
24                 password: await bcrypt.hash(password, 10),
25                 address: {
26                     streetName: streetName,
27                     subdivision: subdivision,
```

only_app > server > controllers > userController.js > default > registerUser > newUser

```
9   export default {
10     registerUser: async (req, res) => {
18       const newUser = {
25         address: {
27           subdivision: subdivision,
28           number: number
29         }
30       }
31
32       await userModel.create(newUser)
33
34       res.status(200).json({ "msg": "todo bien al crear el usuario" })
35
36     } catch (err) {
37       console.log(err)
38       return res.status(500).json({ "msg": "problema de servidor" })
39     }
40   },
41   editUser: async (req, res) => {
42     try {
43
44       const id_user = req.query._id
45       console.log(id_user)
46       const user = await userModel.findById(id_user)
47       if (!user) return res.status(400).json({ "msg": "no hay usuario" })
48
49       user.name = req.body.name ? req.body.name : user.name
```

```
       user.apePat = req.body.apePat ? req.body.apePat : user.apePat
       user.apeMat = req.body.apeMat ? req.body.apeMat : user.apeMat
       user.email = req.body.email ? req.body.email : user.email
       user.numberPone = req.body.numberPhone ? req.body.numberPhone : user.numberPhone
       user.password = req.body.password ? await bcrypt.hash(req.body.password, 10) : user.password
       user.address = req.body.address ? {
         streetName: req.body.address.streetName || user.address.streetName,
         subdivision: req.body.address.subdivision || user.address.subdivision,
         number: req.body.address.number || user.address.number
       } : user.address;

       await userModel.findByIdAndUpdate(id_user, user)
       res.status(200).json({ "msg": "actualizado" })

     } catch (err) {
       console.log(err)
       return res.status(500).json({ "msg": "problema de servidor" })
     }
   },
   getUsers: async (req, res) => {
     try {
       const data = await userModel.find()
```

```

editUser: async (req, res) => {
},
getUsers: async (req, res) => {
  try {
    const data = await userModel.find()
    return res.status(200).send(data)
  } catch (err) {
    return res.status(500).json({ "msg": "problema de servidor" })
  }
},
getUser: async (req, res) => {
  try {
    const id_user = req.query._id
    const user = await userModel.findOne({ _id: id_user })

    return res.status(200).json(user)
  } catch (err) {
    return res.status(500).json({ "msg": "problema de servidor" })
  }
},
login: async (req, res) => {
  try {

```

```

},
login: async (req, res) => {
  try {
    const { email, password } = req.body
    if (!email || !password) return res.status(400).json({ "msg": "credenciales invalidas" })

    const user = await userModel.findOne({ email })
    if (!user) return res.status(400).json({ "msg": "no hay usuario con este correo" })

    if (!await bcrypt.compare(password, user.password)) return res.status(400).json({ "msg": "contraseñas no coinciden" })

    const load = { _id: user._id, email: user.email }
    const token = await jwt.sign(load, process.env.private_key)

    const session = {
      userId: user._id,
      jwt: token
    }

    await sessionModel.create(session)

    return res.status(200).json({ "msg": "login exitoso", token })
  } catch (err) {

```

En esta parte se agregan los controladores enfocados al usuario para poder guardar los datos de los usuarios, agregando la capa de seguridad, ya que se guardarán las contraseñas para poder validar su inicio de sesión.

```

7
8
9 },
10 addContact: async (req, res) => {
11   try {
12     const id_user = req.query._id
13     const user = await userModel.findById( id_user )
14     if (!user) return res.status(400).json({ "msg": "no se encuentra al usuario" })
15
16     const { name, email, numberPhone } = req.body
17     if (!name || !email || !numberPhone) return res.status(400).json({ "msg": "error con una de las en"
18
19     const emerContact = {
20       name: name,
21       email: email,
22       numberPhone: numberPhone
23     }
24
25     await userModel.findByIdAndUpdate(id_user, {
26       $push: {
27         "emerContact": emerContact
28       }
29     })
30

```

En este apartado se agrega lo del número de emergencia, se puede agregar uno o varios contactos de emergencia, esto se hace a partir de la validación del usuario, una vez que pasa esto

Entregable #6

Se identificaron varias fallas durante el proceso de conexión al intentar localizar las zonas en el mapa. Este inconveniente surgió al momento de realizar la búsqueda, lo que generó dificultades para acceder de manera eficiente a la información precisa sobre las áreas geográficas requeridas. Los errores presentados no solo afectaron la rapidez con la que se podía realizar la búsqueda, sino que también comprometieron la exactitud de los resultados obtenidos, dificultando el análisis y la toma de decisiones basada en los datos geoespaciales. Estas fallas podrían estar relacionadas con problemas técnicos en la plataforma de búsqueda o con errores en el sistema de mapeo utilizado, lo cual debe ser revisado y solucionado para evitar futuros contratiempos.

```

server > controllers > zoneController.js > ...
1  import { zoneModel } from "../models/zoneModel.js"
2  import mongoose from "mongoose"
3
4  export default {
5    reportZone: async (req, res) => {
6      try {
7
8        const { dangerLevel, description, coordinates } = req.body
9        if (!dangerLevel || !description || !coordinates) return res.status(400).json({ "msg": "falta un da
10
11        const newReport = {
12          dangerLevel: req.body.dangerLevel,
13          description: req.body.description,
14          location: {
15            coordinates: req.body.coordinates
16          }
17        }
18
19        await zoneModel.create(newReport)
20        return res.status(200).json({ "msg": "se levanto el reporte" })
21      } catch (err) {
22        return res.status(500)({ "msg": "problema de servidor" })
23      }
24    },
25    getZones: async (req, res) => {
26

```

```

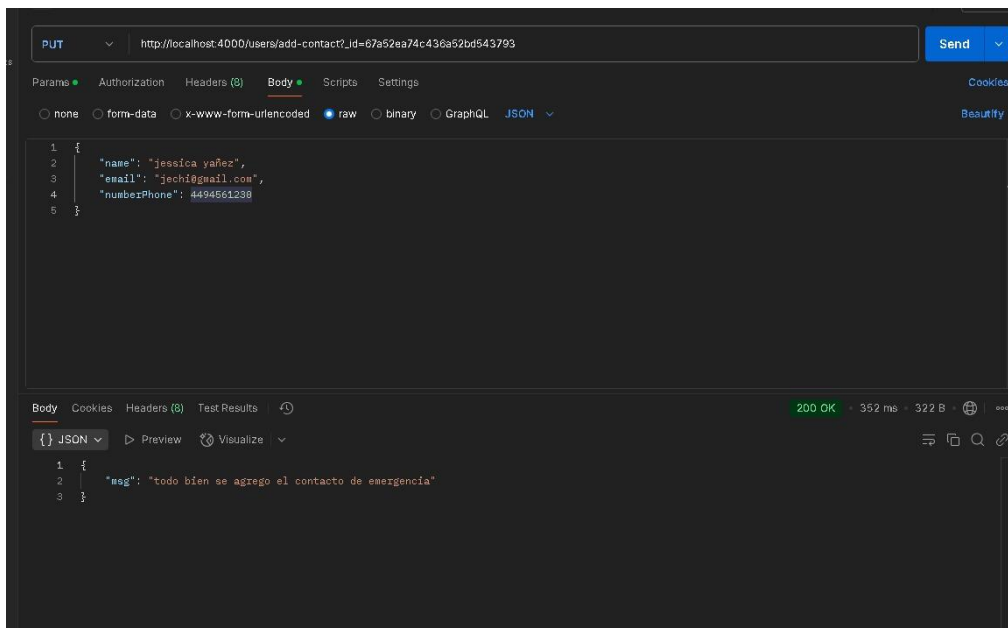
zoneController.js X
server > controllers > zoneController.js > ...
4  export default {
5    reportZone: async (req, res) => {
19      await zoneModel.create(newReport)
20      return res.status(200).json({ "msg": "se levanto el reporte" })
21
22    } catch (err) {
23      return res.status(500)({ "msg": "problema de servidor" })
24    }
25  },
26  getZones: async (req, res) => {
27    try {
28
29      const zones = await zoneModel.find()
30      res.status(200).json(zones)
31
32    } catch (err) {
33      return res.status(500)({ "msg": "problema de servidor" })
34    }
35  }
36 }

```

Entregable #7:

Se procedió a realizar la corrección de los endpoints utilizados para la localización, lo que permitió resolver diversos problemas de conectividad y precisión que se

habían presentado anteriormente. Gracias a estas actualizaciones, los endpoints ahora están completamente alineados con el funcionamiento esperado, lo que asegura una mayor eficiencia en el proceso de localización y mejora la experiencia del usuario al interactuar con la aplicación. Además, se implementaron nuevas zonas en el mapa de nuestra aplicación, lo que amplía la cobertura geográfica disponible y proporciona a los usuarios más opciones y mayor detalle en la visualización de áreas específicas. Estas mejoras fueron fundamentales para optimizar el rendimiento y garantizar una experiencia más fluida y precisa al momento de interactuar con el mapa y la localización dentro de la plataforma.



http://localhost:4000/users/get-one?id=67a52ea74c436a52bd543793

GET http://localhost:4000/users/get-one?id=67a52ea74c436a52bd543793 Send

Params Authorization Headers (0) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (0) Test Results 200 OK · 110 ms · 722 B

```
1 {
2   "address": {
3     "streetName": "tiliches",
4     "subdivision": "tlacuahas",
5     "number": 200
6   },
7   "_id": "67a52ea74c436a52bd543793",
8   "name": "moon",
9   "apePat": "talk",
10  "apeMat": "bebop",
11  "email": "moon@gmail.com",
12  "password": "$2b$10$PMDzqrU1sZRW6R1zSM_jun/VDFk6c6lwQREBa9H0e5oFKMXGw:6a",
13  "numberPhone": "449889911",
14  "_v": 0,
15  "emerContact": [
16    {
17      "name": "mayoneso",
18      "email": "mayo@gmail.com",
19    }
20  ]
21 }
```

Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

http://localhost:4000/users/edit?id=67a52ea74c436a52bd543793

PUT http://localhost:4000/users/edit?id=67a52ea74c436a52bd543793 Send

Params Authorization Headers (0) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "name": "moon",
3   "apePat": "talk",
4   "apeMat": "beat",
5   "email": "moon@gmail.com",
6   "numberPhone": "449889911",
7   "password": "moon",
8   "address": {
9     "streetName": "tiliches",
10    "subdivision": "tlacuahas",
11    "number": 200
12  }
13 }
```

Body Cookies Headers (0) Test Results 200 OK · 332 ms · 288 B

```
1 {
2   "msg": "actualizado"
3 }
```

http://localhost:4000/users/add

SaveShare

POST

http://localhost:4000/users/add

Send

Params

Authorization

Headers (8)

Body

Scripts

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```
1 {
2   "name": "jechi",
3   "apePat": "Rafer",
4   "apeMat": "esparza",
5   "email": "Rafer@gmail.com",
6   "numberPhone": 4498889711,
7   "password": "megustalafanta",
8   "address": {
9     "streetName": "octavio paz",
10    "subdivision": "poetas",
11    "number": 200
12  }
13 }
```

Body

Cookies

Headers (8)

Test Results

200 OK

315 ms

306 B

Visualize

{ } JSON

Preview

Visualize

```
1 {
2   "msg": "todo bien al crear el usuario"
3 }
```