Trabajo Práctico Final: GEA

Diseño de Base de Datos II
Diseño e implementación de un caso real de aplicación de bases de datos NoSQL con MongoDB

Alumnos:

- Marotta, Alejandro Adrián
  - Legajo: 59103
- Soria Gava, Lucas Damián
  - Legajo: 58156

Índice:

¿Qué es GEA?

En los principios de los tiempos reinaba el Caos, donde todo era vacío, oscuridad y confusión. Entonces surgió Gea, diosa primigenia. De la madre Tierra surgió el cielo, las montañas, los llanos, los mares y los ríos. Ella vino a traer orden y claridad al mundo.

Preocupados por las problemáticas ambientales presentes en el mundo actual, decidimos crear un ambiente de visualización y colaboración para darle voz a nuestro planeta.

La meta de nuestra comunidad es evidenciar las problemáticas actuales e impulsar el cambio para mejorar el cuidado ambiental, al mismo tiempo que mostramos la belleza de nuestro mundo.

Buscamos que nuestros usuarios se enamoren del planeta. Creemos que si aprendemos a quererlo, aprenderemos a cuidarlo.

Experiencia de usuario:

En nuestra aplicación los usuarios podrán compartir tanto paisajes bellos como lugares en donde la acción del ser humano ha dañado la belleza del mismo. Para lograrlo planteamos una solución la cual posee dos apartados, el apartado “Beauty” y el apartado “Issues”.

El apartado “Beauty” estará destinado a compartir imágenes que provoquen que nuestros usuarios se sientan más cercanos al planeta. La misma contará con una ubicación, una descripción, “tags” para poder buscar temas relacionados, y la interacción con otros usuarios se realizará a partir de comentarios en la foto, “likes” y “dislikes”.

El apartado de “Issues” se compartirán imágenes que evidencian una problemática y busca generar en el usuario una actitud proactiva hacia el cuidado del planeta. Para ello hemos creado un nuevo concepto llamado “support”, en el cual los usuarios mostraran su apoyo hacia la idea de cambiar aquello que provoque el conflicto.

Los usuarios podrán seguir a otros, de esta manera estarán al tanto de los temas que más les interesan. Para mantener una experiencia afín a nuestro objetivo, hemos decidido que en el perfil de los mismos no se mostrará la cantidad de seguidores que posee pero sí a quienes sigue.

Modelo de negocio:

Si bien la finalidad de nuestra empresa no es el lucro, necesitamos dinero para mantener a nuestro sistema funcionando. Para ello, empresas que estén comprometidas con el medio ambiente, podrán crearse un usuario empresarial en el cual compartirán publicidades.

La misma está enfocada a mostrar las acciones de las organizaciones, para demostrar su cambio de paradigma de negocio y la forma en que toman en cuenta el impacto ambiental que generan, como por ejemplo las empresas de triple impacto o de tipo B. Además, estas publicidades tienen el objetivo de promover a otras industrias a sumarse a este movimiento.

Necesidades del usuario y decisiones de la empresa:

- El usuario debe poder crear un perfil o iniciar sesión.
- Los usuarios deben poder seguir a otros usuarios.
- Deben poder subir fotos, que tengan una descripción, tags y ubicación.
- Las fotos pueden ir en cualquiera de las dos secciones (beauty e issues).
- Deben poder comentar las imágenes, no pueden ser anónimas.
- Los usuarios deben poder poner like y dislike a la publicación beauty.
- Los usuarios deben poder poner support a la publicación issues.
- Los usuarios no podrán consultar quienes son sus seguidores, solamente podrán ver a quienes ellos sigan.
- Todos los perfiles son públicos.
- A cada usuario le aparece en la página principal las publicaciones de las cuentas a las que sigue.
- Las publicaciones realizadas por las empresas se llaman publicidad.
- La publicidad no tiene comentarios.
- Los usuarios deben poder buscar a otros usuarios ó a tags.

Necesidades de la empresa:

En nuestra aplicación GEA priorizamos el acceso rápido a la información de imágenes y usuarios, ya que en torno a estos se basa nuestra experiencia, los clientes que utilicen GEA se verán interesados en ver imágenes o perfiles de forma constante.

GEA también necesita poder tener un control sobre las estadísticas de la aplicación, por lo que es de utilidad poder conocer la cantidad de usuarios y empresas que utilizan el servicio activamente (han publicado en los últimos 6 meses) y cuántos de ellos se han unido a la comunidad en un periodo de tiempo determinado.

También necesitamos saber la cantidad de publicidades que se han publicado, para poder estimar los ingresos, basándose en un precio fijo por publicidad.

Diseño:

¿Por qué se eligieron esas colecciones y datos?

Dadas las condiciones, necesidades y decisiones expuestas en los apartados anteriores, y tomando en cuenta que se utilizara la base de datos NoSQL MongoDB para la implementación, se decidió que las colecciones que la compondrán seran: usuario, beauty, issue y por ultimo add.

Los documentos de la colección “usuario” deben contener nick de usuario, el cual debe ser único, por lo que será el atributo _id del documento y una contraseña. Por razones de contacto y seguridad, se requiere conocer el mail y teléfono del usuario que se está registrando. También se almacenan datos como nombre real, una descripción que represente el pensamiento del usuario y una foto de perfil, los cuales tienen la función de presentar al usuario frente a los demás. A la empresa le interesa conocer la fecha en la que se creó el usuario, por lo que esta también será almacenada.

De cara al usuario y a estrategias empresariales, es también útil llevar un registro de las publicaciones a las que se les dió like, dislike o support, así como también es importante llevar registro de las personas que el usuario decide seguir. Este tipo de registro se lleva a cabo a través de una lista que irá creciendo a medida que el usuario utilice la aplicación.
A diferencia de los usuarios, las empresas no tiene permitido seguir otras cuentas, dar likes, dislikes o supports, por lo que estos campos no se almacenarán en aquellas cuentas que tengan el campo “empresa” como true.

Para almacenar las publicaciones, se tiene presente que los aspectos más importantes que debe tener el documento son: id del usuario que la crea y una imagen. La fecha en que se realiza la publicación es de especial interés, ya que los usuarios buscan estar al tanto de las últimas fotos subidas, especialmente aquellas que pertenecen a la sección de “issue”. Los usuarios también deberían ser capaces de buscar imágenes por tipo, para lograrlo se propone un sistema de “tags”, el cual es una lista con palabras clave que caracterizan a la imagen o problemática planteada. Para poder aportar aún más información a las imágenes publicadas se almacenará también una descripción y la ubicación.

Para que los usuarios puedan reaccionar y comunicarse entre ellos, se propone la posibilidad de poder comentar las imágenes o dar “like” / “dislike” ó “support”. En el caso de las imágenes de la sección “beauty” se obtendrá el sistema de likes y dislikes, para expresar una medida de “belleza” del lugar. Mientras que en la sección de “issue” se podrá reaccionar mediante “supports”, los cuales muestran el apoyo y unión a la causa o problemática que se plantea.

Los comentarios que se pueden realizar en estas publicaciones deben contener dos datos: el usuario que redactó el mensaje y el comentario en cuestión.

Aunque se podían mantener los dos tipos de publicaciones bajo la misma colección, se decidió que se separarían ambas en dos porque se busca alta velocidad en las consultas realizadas sobre estas dos categorías. Inicialmente se había planteado una solución donde ambas estaban unidas bajo la colección “publicaciones” y se planteó diferenciar ambas por la presencia de un atributo binario llamado “issue”, similar al sistema “empresa” utilizado en los usuarios. El problema con esta solución es que la aplicación divide en dos secciones su página principal, una sección de beauty y otra de issue, donde se puede encontrar en cada una imágenes exclusivas de esa sección. Esto lleva al problema de que una consulta debe ir revisando cada una de las publicaciones para comprobar a qué sección pertenece. Esta acción toma mucho tiempo y es por lo tanto va en contra del deseo de velocidad de consultas expresado anteriormente. Planteando una solución con índices se llega a un problema similar, dado que aproximadamente la mitad de las imágenes pertenecen a una colección y la otra mitad a la segunda, el uso de índices es antiproducente, ya que se recomienda que un valor del índice no contenga más de un 30% de los datos frente a una consulta.

A diferencia de las colecciones anteriores, los documentos almacenados en la colección de publicidades poseen una menor cantidad de datos y no permiten realizar comentarios. Esta última decisión se tomó para evitar expresiones de odio hacia las empresas responsables de dichas publicidades. Sin embargo, se permite el uso del sistema de “likes” y “dislikes”, para que la empresa sepa cómo son recibidas sus acciones por la comunidad de GEA, además de que sirve a los usuarios como medio de expresión.

Las mismas por lo tanto tendrán: información de la empresa que la publicó, fecha de publicación, descripción, tags y una imagen.

¿Cómo se plantea el funcionamiento de la aplicación?

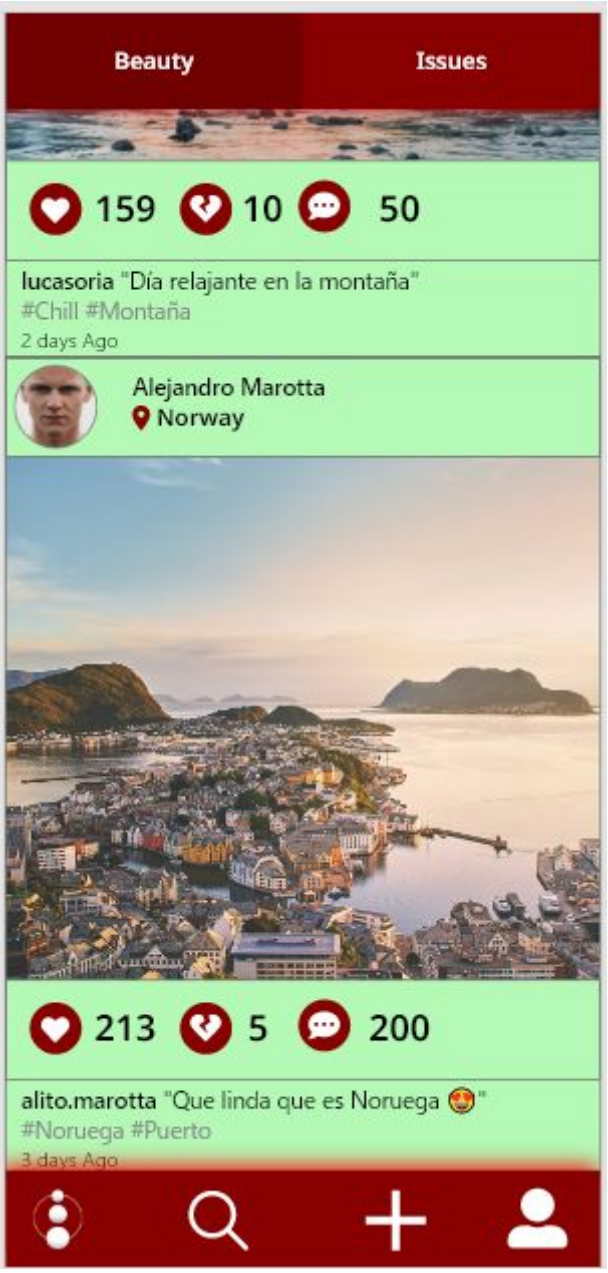
La aplicación está pensada para que la primera pantalla que se presenta al usuario cuando éste instala la aplicación, sea la página de “login”.

Esta sería la primera interacción entre el usuario y la base de datos. La aplicación deberá consultar que la información provista por el usuario es correcta y permitir el inicio de sesión o no en base al resultado.

En caso de que el usuario no posea un usuario, este podrá registrarse presionando el botón correspondiente.

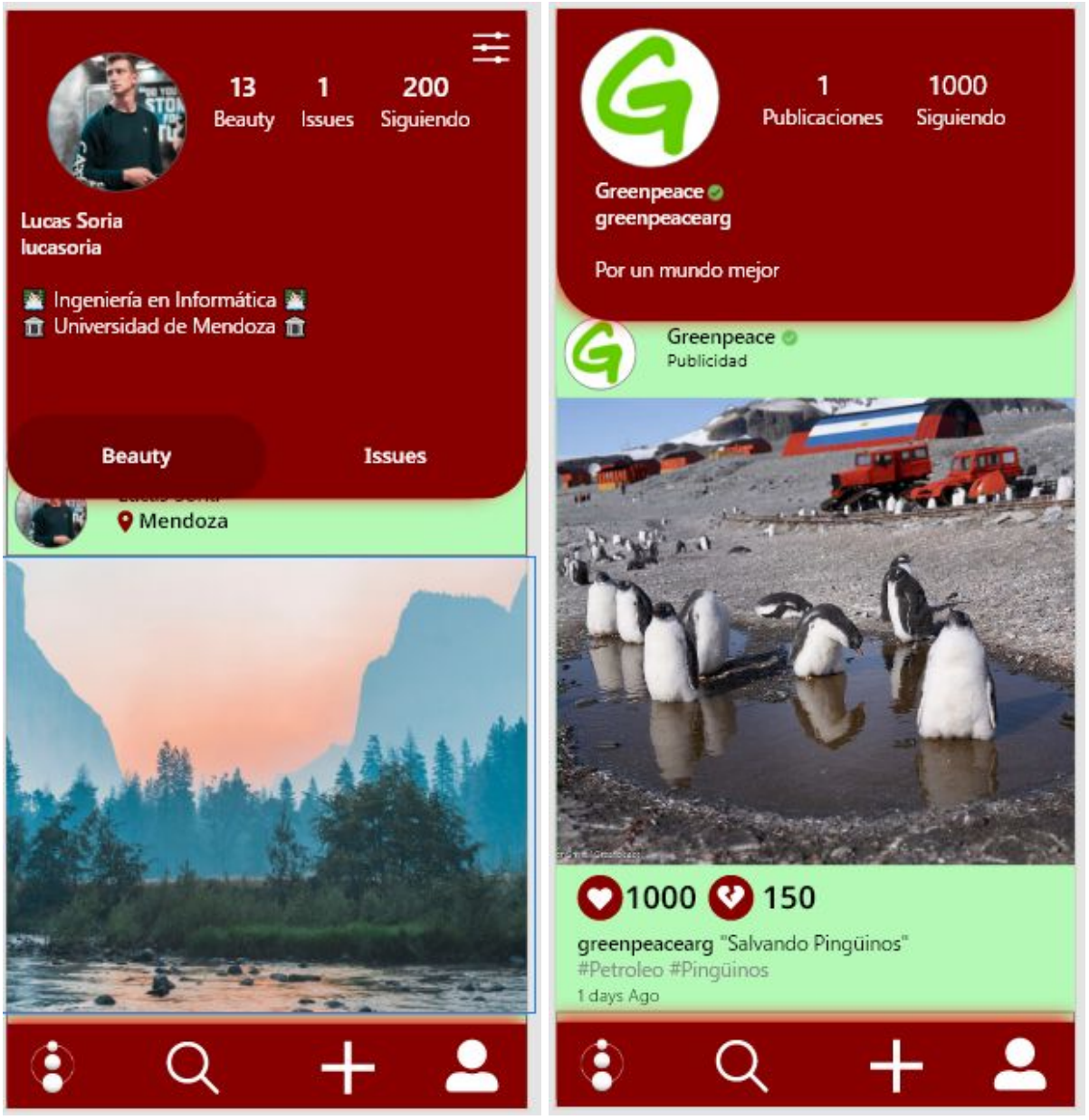
En esta pantalla el usuario deberá ingresar los datos solicitados, los cuales se verificarán antes de ser ingresados en la base de datos. En esta ocasión se deberá comprobar que el usuario no se encuentre ya registrado en la base de datos.

Una vez que el usuario esté registrado o haya iniciado sesión, se presentará la pantalla principal o “feed” del usuario, la cual está dividida en 2, cada división mostrando una sección. Por defecto se enseña la sección de beauty. El “feed” presenta al usuario las últimas publicaciones de las cuentas a las que sigue el usuario y entre ellas, publicidades.



Presionando en el botón correspondiente se puede cambiar a la sección de “issue”, donde se presentarán las imágenes correspondientes, siguiendo el mismo criterio que en “beauty”.

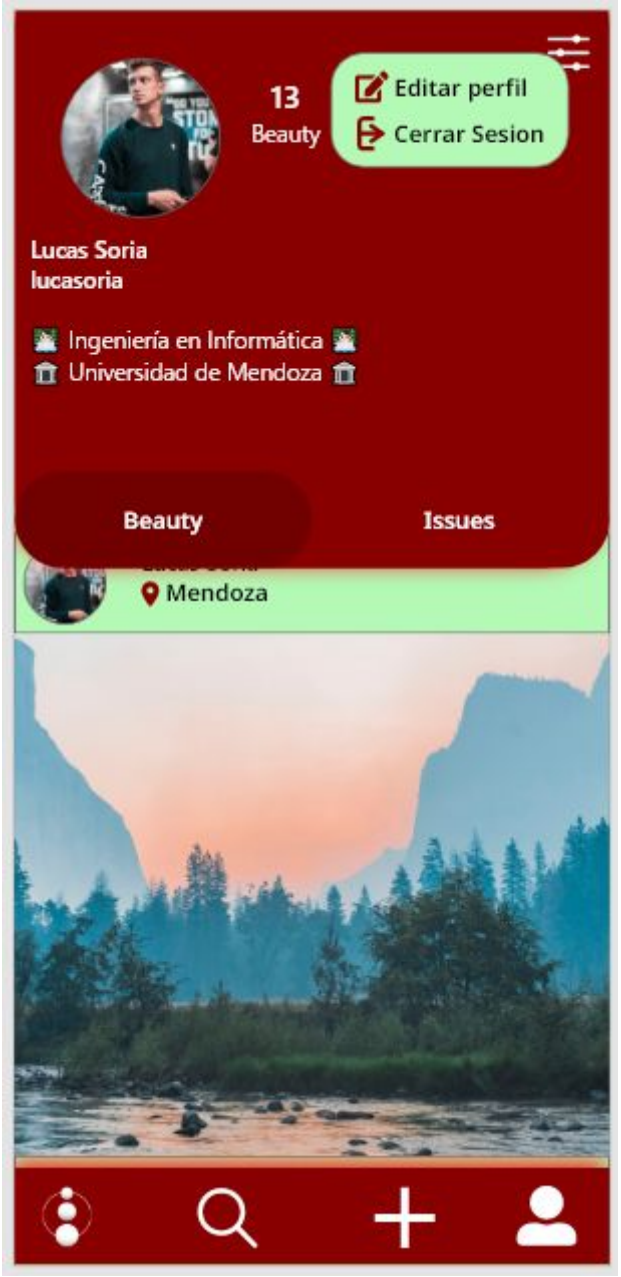
De la misma forma, se puede ingresar al apartado de usuario (normal o empresa), mostrando la información relevante para este.



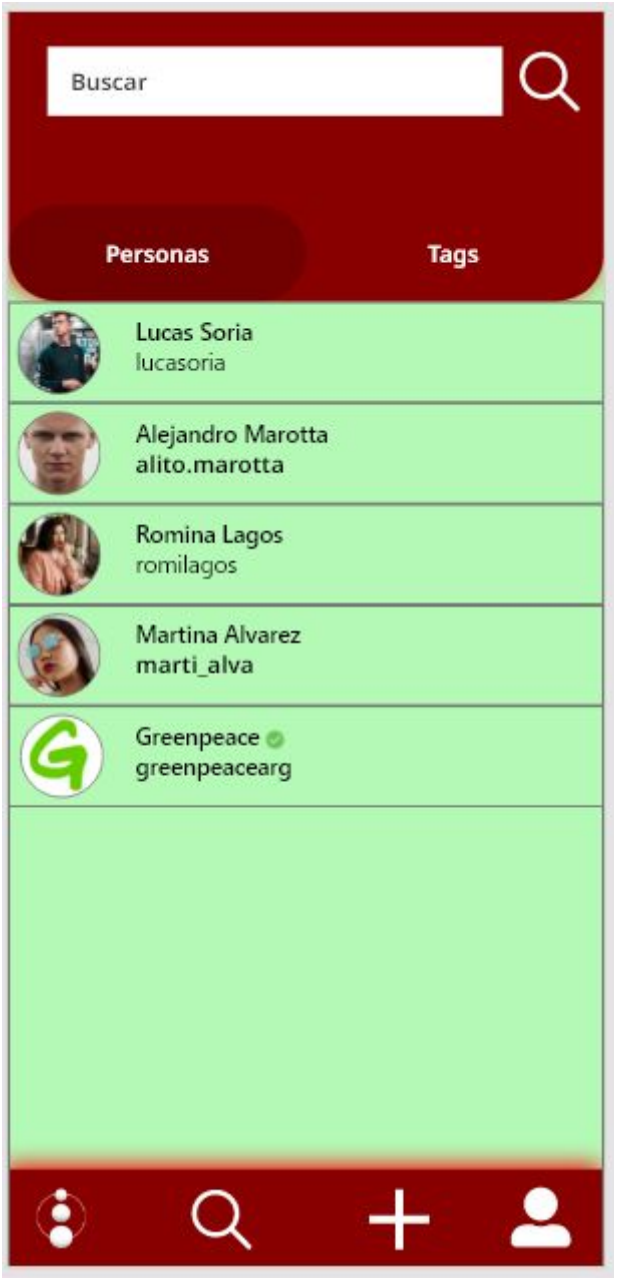
Este apartado muestra la cantidad de publicaciones “beauty” o “issue” que el usuario realizó, así como la cantidad de personas a las que sigue. Muestra su nombre, nombre de usuario y su descripción, así como las publicaciones que hizo el mismo separadas en las secciones, en orden inverso de publicación.

En el caso de ser un perfil de empresa se verán la cantidad de publicaciones que realizó, la cantidad de usuarios a los que sigue, una descripción y tendrán un símbolo de verificación junto a su nombre.

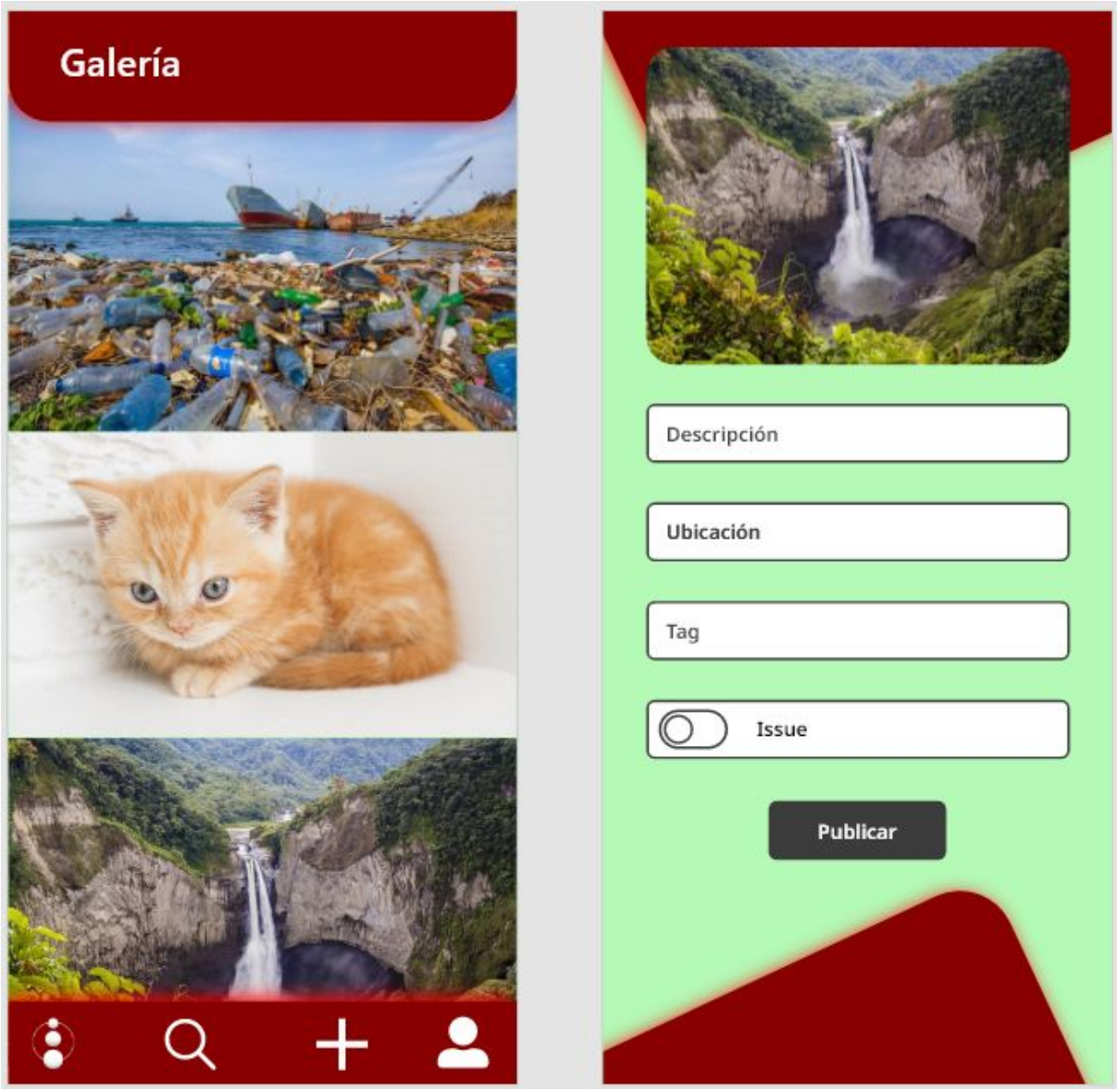
También presenta la posibilidad de, mediante un menú, modificar los datos del usuario o de cerrar sesión, según decida el usuario.



Otra característica presente en la aplicación es la búsqueda de usuarios y de fotos por “tag”. Ambas opciones están presentes en la sección de búsqueda.



Otra de las acciones que debe ser capaz de hacer el usuario es la de publicar una imagen. Esto se puede hacer mediante el apartado correspondiente. Pudiendo allí seleccionar una imagen y agregarle su descripción y ubicación.



Colecciones:

Usuario:

{
“_id”: “juanpedrito33”,
“contraseña” : “boquitaelmasgrande”,
“nombre”: “Juan Pedro”,
“mail”: “jpedroelmejor@yahoo.com”,
“teléfono”: 2615489346,
“descripción”: “·32 vueltas al sol”,
“foto_perfil”: “url”,
“empresa”: false,
“fecha_creacion”: new Date(),
“seguidos”: [<ObjectIDUsuario>, ….],
“likes”: [<ObjectIDPublicacion>, ….],
“dislikes”: [<ObjectIDPublicacion>, ….],
“support”: [<ObjectIDPublicacion>, ….],
},
{
“_id”: “b&mFoundation”,
“contraseña” : “billymelinda4ever”,
“nombre”: “Bill & Melinda Gates Foundation”,
“mail”: “b&m4ever@microsoft.com”,
“teléfono”: 2615489346,
“descripción”: “Fundación de Bill & Melinda. Ayudamos al mundo a ser un lugar mejor”,
“foto_perfil”: “url”,
“empresa”: true,
“fecha_creacion”: new Date(),
}

Publicación beauty:

{
“_id”: 981u29,
“id_usuario”: <ObjectIDUsuario>,
“fecha_publicacion”: new Date(),
“foto”: “url”,
“Ubicación”: {
“latitud”: -40.761666
“longitud”: -71.645209
“nombre”: “Villa La Angostura, Neuquén”
}
“descripción”: “Miren este hermosos amanecer”,
“tag”: [“Montaña”,”Amanecer”],
“likes”: [<ObjectIDUsuario>, ….],
“dislikes”: [<ObjectIDUsuario>, ….],
“comentarios”:
[{
“userid”: <ObjectIDUsuario1>,
“comentario”: “Hermoso ❤😍”,
},
{
“userid”: <ObjectIDUsuario2>,
“comentario”: “Quiero irrr 😭”
},]
}

Publicación Issue:

{
“_id”: 981u30,
“id_usuario”: <ObjectIDUsuario>,
“fecha_publicacion”: new Date()
“foto”: “url”,
“descripción”: “Antes podía ver la cordillera, ahora no puedo ver nada por culpa de la contaminación!!!”,
“tag”: [“Basura”,”Contaminación”],
“Ubicación”: {
“latitud”: -32.881312
“longitud”: -68.860408
“nombre”: “ Ciudad de Mendoza, Mendoza”,
}
“support”: [<ObjectIDUsuario>, ….],
“comentarios”:
[{
“userid”: <ObjectIDUsuario1>,
“comentario”: “Hay que hacer leyes de regulación de contaminación!!!”,
},
{
“userid”: <ObjectIDUsuario2>,
“comentario”: “Empecemos a usar bicis!!”
}]
}

Publicidad:

{
“_id”: 981u31,
“id_usuario”: <ObjectIDUsuario>,
“fecha_publicacion”: new Date()
“foto”: “url”,
“descripción”: “Con Melinda & Bill brindamos agua a pueblos de África”,
“tag”: [“Basura”,”Contaminación”],
“likes”: [<ObjectIDUsuario>, ….],
“dislikes”: [<ObjectIDUsuario>, ….]
}

Indexaciones:

Índice de publicaciones por id de usuario:

```
db.beauty.createIndex( { id_usuario: 1 }, { background: true } );
db.issues.createIndex( { id_usuario: 1 }, { background: true } );
```

Índice de publicaciones por id de empresa:

```
db.adds.createIndex( { id_usuario: 1 }, { background: true } );
```

Índice por tags:

```
db.beauty.createIndex( { tag: 1 } , { background: true } );
db.issues.createIndex( { tag: 1 }, { background: true } );
db.adds.createIndex( { tag: 1 }, { background: true } );
```

Documentación y consultas:

Crear base de datos:

```
use gea;
```

Crear la colección:

```
db.createCollection(“users”);
db.createCollection(“beauty”);
db.createCollection(“issues”);
db.createCollection(“adds”);
```

Log in:

```
db.system.js.save({
        _id: "logIn",
        value: function(id_usuario, contra) {
                usuario = db.users.findOne( { _id: id_usuario }, { contrasena: 1 } )
                if (usuario != null) {
                        if (usuario.contrasena == contra) { return true; }
                        else { return false; }
                } else { return false; }
        }
});
```

Registrar un usuario:

```
db.system.js.save({
        _id: "createUser",
        value: function(id_usuario, con, nom, email, tel, emp) {
                usuario = db.users.findOne( { _id: id_usuario }, { _id: 1 } );
                if (usuario == null) {
                        db.users.insert({ _id: id_usuario, contrasena: con, nombre: nom, mail:
                email, telefono: tel, foto_perfil: "perfiles/default.jpg", empresa: emp,
                fecha_creacion: new Date() });
                } else { return false }
        }
});
```

Ejemplo:

db.loadServerScripts();
createUser(“juanchi”, “hurlingClub753”, “Juan Pedro de la Rosa”, “jrosa@gmail.com”, “2616958433”, ”La naturaleza necesita nuestra voz”, false);

Insertar un post en Beauty:

db.beauty.insert({ id_usuario: "lucasoria", fecha_publicacion: new Date(), foto: "post/beauty/lucasoria/montana.jpg", descripcion: "Hermosa vista de la montaña", tag: ["montana", "rio"], ubicacion: { latitud: 30000, longitud: -24685, nombre: "Cordillera" } });

Insertar un post en Issue:

db.issues.insert({ id_usuario: "lucasoria", fecha_publicacion: new Date(), foto: "post/issue/lucasoria/contaminacion.jpg", descripcion: "Antes podía ver la cordillera, ahora no puedo ver nada por culpa de la contaminación!!!", tag: ["basura", "contaminacion"], ubicacion: { latitud: -32.881312, longitud: -68.860408, nombre: "ciudad de Mendoza, Mendoza" } });

Traer todos los Beauty en tu Feed:

```
db.system.js.save({
	_id: "seePostsBeauty",
	value: function(userid, limit) {
		var lista = [];
		seguidos = db.users.findOne( { _id: userid }, { seguidos: 1 } ).seguidos;
		for (var i = 0; i < db.beauty.find( { id_usuario: { $in: seguidos } }, { _id: 1 }
	).count(); i++) {
			docs = db.beauty.find( { id_usuario: { $in: seguidos } } ).sort( {
		fecha_publicacion: -1 } ).limit(limit).skip(i*limit).toArray();
			if (docs.length >= 1) {
				lista.push(docs);
			}
		}
		return lista;
	}
});
```

Traer todos los Issue en tu feed:

```
db.system.js.save({
	_id: "seePostsIssues",
	value: function(userid, limit) {
		lista = [];
		seguidos = db.users.findOne( { _id: userid }, { seguidos: 1 } ).seguidos;
		for (var i = 0; i < db.issues.find( { id_usuario: { $in: seguidos } }, { _id: 1 }
	).count(); i++) {
			docs = db.issues.find( { id_usuario: { $in: seguidos } } ).sort( {
		fecha_publicacion: -1} ).limit(limit).skip(i*limit).toArray();
			if (docs.length >= 1) {
				lista.push(docs);
			}
		}
		return lista;
	}
});
```

Traer el perfil de la persona o empresa:

```
db.system.js.save({
	_id: "seeProfile",
	value: function(userid) {
		return db.users.findOne( { _id: userid } );
	}
});
```

En el apartado de buscar:

Buscar coincidencias en Usuarios:

```
db.system.js.save({
	_id: "findUsers",
	value: function(id_usuario) {
		usuarios = db.users.find( { _id: { $regex: '^' + id_usuario, $options: "$i" } }, {
	nombre: 1 } );
		if (usuarios.count() == 0) {
			return false;
		} else { return usuarios; }
	}
});
```

Buscar coincidencias en tags:

```
db.system.js.save({
	_id: "findTags",
	value: function(t) {
		lista = [];
		pB = db.beauty.find( { tag: t }, { id_usuario: 1, foto: 1, tag: 1 } ).toArray();
		for (var i = 0; i < pB.length; i++) {
			lista.push(pB[i]);
		}
		pI = db.issues.find( { tag: t }, { id_usuario: 1, foto: 1, tag: 1 } ).toArray();
		for (var i = 0; i < pI.length; i++) {
			lista.push(pI[i]);
		}
		pA = db.adds.find( { tag: t }, { id_usuario: 1, foto: 1, tag: 1 } ).toArray();
		for (var i = 0; i < pA.length; i++) {
			lista.push(pA[i]);
		}
		return lista;
	}
});
```

Editar un perfil:

```
db.system.js.save({
	_id: "editUserDatos",
	value: function(id_usuario, nom, email, tel, des, foto) {
		db.users.update( { _id: id_usuario }, { $set: { nombre: nom, mail: email,
	telefono: tel, descripcion: des, foto_perfil: 'perfiles/'+foto } } );
	}
});
```

Editar contraseña:

```
db.system.js.save({
        _id: "editUserContrasena",
        value: function(id_usuario, con_antigua, con_nueva) {
                usuario = db.users.findOne( { _id: id_usuario } );
                if (usuario.contrasena == con_antigua) {
                        db.users.update( { _id: id_usuario }, { $set: { contrasena: con_nueva }
                } );
                } else { return false; }
        }
});
```

Cambiar de usuario a empresa:

```
db.system.js.save({
        _id: "editUserEmpresa",
        value: function(id_usuario, emp) {
                db.users.update( { _id: id_usuario }, { $set: { empresa: emp } } );
        }
});
```

Comentar un Issue o Beauty:

```
db.system.js.save({
        _id: "coment",
        value: function(seccion, publicacion, userid, comentario) {
                if (seccion == "beauty") {
                        db.beauty.findOneAndUpdate( { _id: publicacion }, { $push: {
                comentarios: { userid: userid, comentario: comentario } } } );
                } else {
                        db.issues.findOneAndUpdate( { _id: publicacion }, { $push: {
                comentarios: { userid: userid, comentario: comentario } } } );
                }
        }
});
```

Usuarios que son empresas:

```
empresas = db.users.find( { empresa: true}, { nombre: 1 } );
```

Cantidad de Empresas:

```
empresas = db.users.find( { empresa: true }, { nombre: 1 } ).count();
```

Usuarios que no son empresas:

```
usuarios = db.users.find( { empresa: false }, { nombre: 1 } );
```

Ver las publicaciones de hace un tiempo determinado y su cantidad:

```
db.system.js.save({
	_id: "findPostByTime",
	value: function(fecha) {
		lista = [];
		pB = db.beauty.find( { fecha_publicacion: { $gte: ISODate(fecha) } }
	).toArray();
		for (var i = 0; i < pB.length; i++) {
			lista.push(pB[i]);
		}
		pI = db.issues.find( { fecha_publicacion: { $gte: ISODate(fecha) } } ).toArray();
		for (var i = 0; i < pI.length; i++) {
			lista.push(pI[i]);
		}
		pA = db.adds.find( { fecha_publicacion: { $gte: ISODate(fecha) } } ).toArray();
		for (var i = 0; i < pA.length; i++) {
			lista.push(pA[i]);
		}
		return [lista , lista.length]
	}
});
```

Quienes publicaron en el último "n unidad de tiempo" y cuantas publicaciones hay:

```
db.system.js.save({
	_id: "whoPosted",
	value: function(tiempo) {
		publicacionesBeauty = db.beauty.find( { fecha_publicacion: { $gte: ISODate(tiempo) } }, {id_usuario: 1 } ).toArray();
		publicacionesIssues = db.issues.find( { fecha_publicacion: { $gte: ISODate(tiempo) } }, {id_usuario: 1 } ).toArray();
		publicacionesAdds = db.adds.find( { fecha_publicacion: { $gte: ISODate(tiempo) } }, {id_usuario: 1 } ).toArray();
		var publicadores = [];
		for (var i = 0; i < publicacionesBeauty.length; i++) {
			if (!publicadores.includes(publicacionesBeauty[i].id_usuario)) {
				publicadores.push(publicacionesBeauty[i].id_usuario);
			}
		}
		for (var i = 0; i < publicacionesIssues.length; i++) {
			if (!publicadores.includes(publicacionesIssues[i].id_usuario)) {
				publicadores.push(publicacionesIssues[i].id_usuario);
			}
		}
		for (var i = 0; i < publicacionesAdds.length; i++) {
			if (!publicadores.includes(publicacionesAdds[i].id_usuario)) {
				publicadores.push(publicacionesAdds[i].id_usuario);
			}
		}
		var publicacionesTotales = publicacionesBeauty.length + publicacionesIssues.length + publicacionesAdds.length;
		return [publicacionesTotales, publicadores];
	}
} );
```

Likes, Dislikes y Supports:

Like a beauty:

```
db.system.js.save({
        _id: "likeBeauty",
        value: function(id_usuario, publicacion) {
                usuario = db.users.findOne( { _id: id_usuario } );
                likes = db.beauty.findOne( { _id: ObjectId(publicacion) }, {likes: 1} ).likes;
                dislikes = db.beauty.findOne( { _id: ObjectId(publicacion) }, {dislikes: 1}
        ).dislikes;
                if ( !likes.includes(id_usuario) && !dislikes.includes(id_usuario) ) {
                        db.users.update( { _id: id_usuario }, { $push: { likes:
        ObjectId(publicacion) } });
                        db.beauty.update({ _id: ObjectId(publicacion) }, { $push: {
                likes:usuario._id }});
                }
        }
});
```

Dislikes a beauty:

```
db.system.js.save({
        _id: "dislikeBeauty",
        value: function(id_usuario, publicacion) {
                usuario = db.users.findOne( { _id: id_usuario } );
                likes = db.beauty.findOne( { _id: ObjectId(publicacion) }, {likes: 1} ).likes;
                dislikes = db.beauty.findOne( { _id: ObjectId(publicacion) }, {dislikes: 1}
        ).dislikes;
                if ( !likes.includes(id_usuario) && !dislikes.includes(id_usuario) ) {
                        db.users.update( { _id: id_usuario }, { $push: { dislikes:
        ObjectId(publicacion) } } );
                        db.beauty.update( { _id: ObjectId(publicacion) }, { $push: { dislikes:
                usuario._id } } );
                }
        }
});
```

Support a issue:

```
db.system.js.save({
        _id: "support",
        value: function(id_usuario, publicacion) {
                usuario = db.users.findOne( { _id: id_usuario } );
                supp = db.issues.findOne( { _id: ObjectId(publicacion) }, {support: 1}
        ).support;
                if ( !supp.includes(id_usuario) ) {
                        db.users.update( { _id: id_usuario }, { $push: { support:
        ObjectId(publicacion) } } );
                        db.issues.update( { _id: ObjectId(publicacion) }, { $push: { support:
                usuario._id } } );
                }
        }
});
```

Like a Adds:

```
db.system.js.save({
        _id: "likeAdd",
        value: function(id_usuario, publicacion) {
                usuario = db.users.findOne( { _id: id_usuario } );
                likes = db.adds.findOne( { _id: ObjectId(publicacion) }, {likes: 1} ).likes;
                dislikes = db.adds.findOne( { _id: ObjectId(publicacion) }, {dislikes: 1}
        ).dislikes;
                if ( !likes.includes(id_usuario) && !dislikes.includes(id_usuario) ) {
                        db.users.update( { _id: id_usuario }, { $push: { likes:
        ObjectId(publicacion) } } );
                        db.adds.update( { _id: ObjectId(publicacion) }, { $push: { likes:
                usuario._id } } );
                }
        }
});
```

Dislikes a Adds:

```
db.system.js.save({
	_id: "dislikeAdd",
	value: function(id_usuario, publicacion) {
		usuario = db.users.findOne( { _id: id_usuario } );
		likes = db.adds.findOne( { _id: ObjectId(publicacion) }, {likes: 1} ).likes;
		dislikes = db.adds.findOne( { _id: ObjectId(publicacion) }, {dislikes: 1} ).dislikes;
		if ( !likes.includes(id_usuario) && !dislikes.includes(id_usuario) ) {
			db.users.update( { _id: id_usuario }, { $push: { dislikes: ObjectId(publicacion) } } );
			db.adds.update( { _id: ObjectId( publicacion ) }, { $push: { dislikes: usuario._id } } );
		}
	}
});
```

Consulta de likes de un usuario:

```
db.system.js.save({
	_id: "findLikes",
	value: function(id_usuario) {
		likes = db.users.findOne( { _id: id_usuario }, { likes: 1 } ).likes;
		lista = [];
		docs = db.beauty.find( { _id: { $in: likes } }, { id_usuario: 1 , foto: 1 } ).toArray();
		for (var i = 0; i < docs.length; i++) {
			lista.push(docs[i]);
		}
		docs = db.adds.find( { _id: { $in: likes }}, { id_usuario: 1, foto: 1 } ).toArray();
		for (var i = 0; i < docs.length; i++) {
			lista.push(docs[i]);
		}
		return lista;
	}
});
```

Consulta de supports de un usuario:

```
db.system.js.save({
	_id: "findSupports",
	value: function(id_usuario) {
		supports = db.users.findOne( { _id: id_usuario }, { support: 1 } ).support;
		lista = [];
		docs = db.issues.find( { _id: { $in: supports } }, { id_usuario: 1 , foto: 1 }
	).toArray();
		for (var i = 0; i < docs.length; i++) {
			lista.push(docs[i]);
		}
		return lista;
	}
});
```

Usuarios nuevos dentro de la app en el último "n unidad de tiempo":

```
db.system.js.save({
	_id: "newUsers",
	value: function(tiempo) {
		usuarios = db.users.find( { fecha_creacion: { $gte: ISODate(tiempo) } }, { _id:
	1 } ).toArray();
		lista = [];
		for (var i = 0; i < usuarios.length; i++) {
			lista.push(usuarios[i]._id);
		}
		return lista;
	}
});
```

Ganancias aproximadas:

```
db.system.js.save({
	_id: "profit",
	value: function(fecha, coste) {
		cantidad = db.adds.find( { fecha_publicacion: { $gte: ISODate(fecha) } }, { _id:
	1 } ).count();
		return cantidad * coste;
	}
});
```