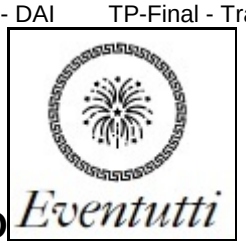


OBJETIVO

Crear una app con React Native (Eventos), utilizando el backend que desarrollamos en el primer cuatrimestre.
Para su desarrollo se debe tener localmente la API de node, utilizar ngrok para generar un túnel y también localmente la app de react native, que se va a probar mediante el uso del dispositivo móvil.

NAVEGACIÓN:

TBD

LA APLICACIÓN:

- Tiene que tener un splash screen y un icono.

USUARIO ANÓNIMO.

- Tiene que utilizar la autenticación para determinar que usuario se está logueado y mostrar la información acorde al mismo.
- Debe registrar nuevos usuarios.

USUARIO AUTENTICADO.

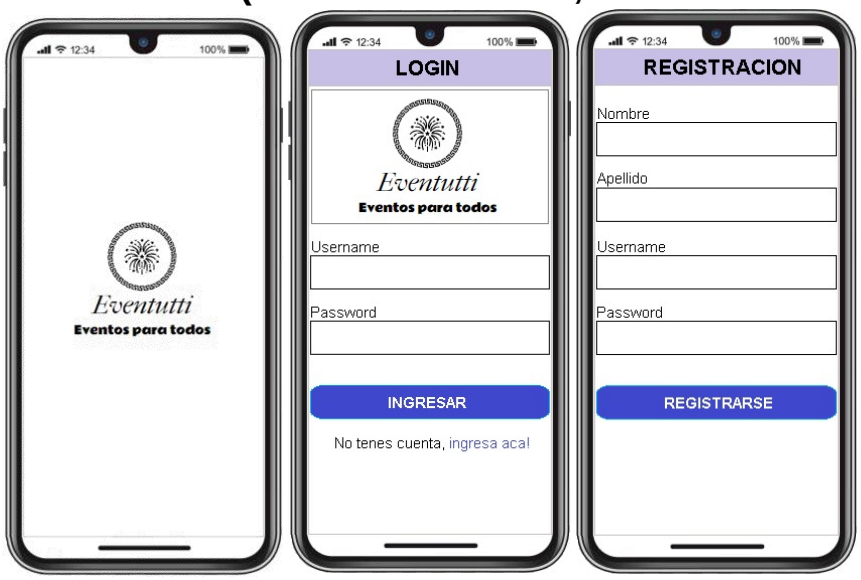
- Debe mostrar tanto los eventos pasados como los próximos (api) y permite presionar en un evento para mostrar el detalle del mismo (api).
- En la pantalla de detalle de un evento además de la información del evento, se tiene que:
    - Para los eventos que ya ocurrieron:
        - Se debe mostrar el listado de los participantes indicando para cada uno si asistió o no. (api)
        - No tengo que tener la posibilidad de suscribirme al evento.
    - Para los eventos próximos:
        - Tengo que poder inscribirme o desuscribirme al evento (se debe validar), y muestra una mensaje en cada caso. (api)
- Debe poseer una pantalla de búsqueda en la que muestra en una lista de todas las categorías de los eventos (api).

  Al presionar un evento de la lista se debe mostrar todos los eventos próximos de esa categoría (api).
  Al presionar un evento se debe mostrar el detalle del mismo (la misma pantalla que en los puntos anteriores)

- Tiene que tener, una pantalla para visualizar el perfil del usuario (solo muestra el nombre y apellido del usuario logueado) (api)
- Tiene que poderse desloguear.
- Cuando el usuario se loguea por primera vez exitosamente, se debe almacenar en el dispositivo la información de inicio de sesión (AsyncStorage), de manera que la próxima vez que se inicie la aplicación no se le pida al usuario sus credenciales.

  Nota: analice también la posibilidad de utilizar SecureStore.

PANTALLAS (de forma anónima, sin identificación):



SplashScreen:

Muestra la imagen de fondo, mientras dura la carga de la app.

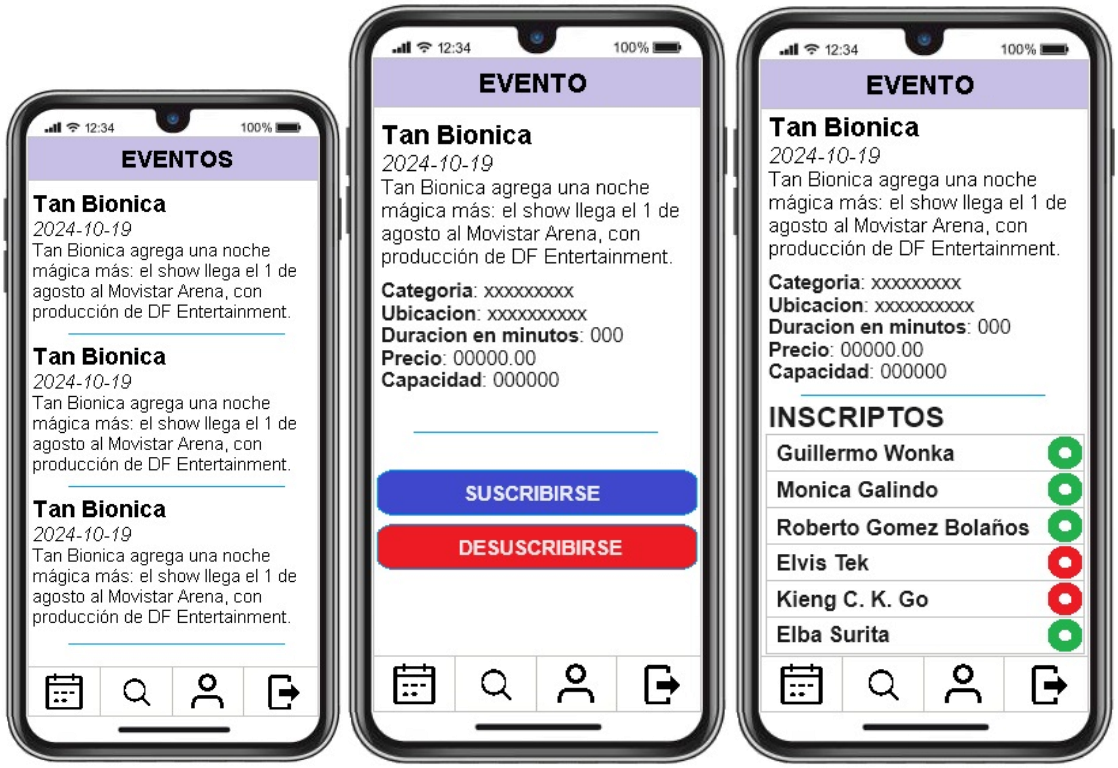
LoginScreen:

Ingresar el UserName y la clave. En caso exitoso navega hacia la pantalla de eventos, y opcionalmente almacena la información en al AsyncStorage / SecureStore, con el objetivo de que la próxima vez no se pida el login.

RegistroScreen:

Permite registrarse a la aplicación. En caso exitoso aparte de agregar el usuario a la base de datos, navega hacia la pantalla de eventos, y opcionalmente almacena la información en al AsyncStorage / SecureStore, con el objetivo de que la próxima vez no se pida el login.

PANTALLAS (con el usuario autenticado):



EventosScreen:

Muestra todos eventos anteriores y futuros, permite scrollear y al presionar un evento, navega a la visualización del mismo.

UnEvento:

Muestra la información del Evento (tanto para eventos anteriores como para eventos futuros).

En el caso de que el evento sea a futuro: Debe permitir la suscripción o la suscripción y muestra un mensaje.

Nota: Nota, no puedo suscribirme más de una vez a un mismo evento o desuscribirme a un evento al que no estoy suscripto. (esto ya se encuentra en la api).

En el caso de que el evento sea anterior: Debe mostrar un listado de los suscriptos y una identificación (roja o verde) indicando que asistió o no.

APIS del TP de Eventos de NODE

Estos son los EndPoints (APIs) del TP Final de Eventos que son utilizados.

- **POST** /api/user/login
- **POST** /api/user/register
- **GET** /api/event/
- **GET** /api/event/**{id}**
- GET /api/event/{id}/enrollment
- **GET** /api/event-category
- **POST** /api/event/**{id}**/enrollment/ **(necesita autenticación)**

- **DELETE** /api/event/**{id}**/enrollment/ **(necesita autenticación)**

Endpoints de mi back:

Users-controller.js:

```
import {Router} from 'express';
import express from "express";
import UsersService from "../services/users-service.js"
import jwt from 'jsonwebtoken';
//import bcrypt from 'bcryptjs';
const UserRouter =  Router();
const svc = new UsersService()

//6
UserRouter.post('/login', async (req, res) => {
    const { username, password } = req.body;
    console.log('username:', username);
    console.log('password:', password);

    try {
        const user = await svc.login(username, password);
        console.log('User retrieved from DB:', user);

        if (user == null) {
            console.log('User not found');
            return res.status(401).json({
                success: false,
                message: 'Usuario o clave inválida.',
                token: ''
            });
        }

        console.log('OK',user);
        const token = jwt.sign(user, 'your_jwt_secret', { expiresIn: '1h'
});
        console.log('Generated token:', token);

        res.status(200).json({
            success: true,
            message: 'Login exitoso.',
            token: token
        });
    } catch (error) {
        console.error('Error during login:', error);
        res.status(500).json({ success: false, message: error.message });
    }
});
```

```
// 6.2 NO FUNCIONA
UserRouter.post('/register', async (req, res) => {
    const { id, first_name, last_name, username, password } = req.body;

    if (!first_name || !last_name) {
        return res.status(400).json({ message: 'Los campos first_name o
last_name están vacíos.' });
    }

    const emailRegex = /\S+@\S+\.\S+/;
    if (!emailRegex.test(username)) {
        return res.status(400).json({ message: 'El email (username) es
sintácticamente inválido.' });
    }

    if (password.length < 3) {
        return res.status(400).json({ message: 'El campo password tiene
menos de 3 letras.' });
    }

    try {
       // const hashedPassword = await bcrypt.hash(password, 10);
        const newUser = await svc.crearUser({ id, first_name, last_name,
username, password});
        console.log('New user created:', newUser);
        res.status(201).json({ message: 'Usuario registrado
exitosamente.' });
    } catch (error) {
        console.error('Error during registration:', error);
        res.status(500).json({ message: error.message });
    }
});
export default UserRouter;
```

events-controller.js:

```
import {Router} from 'express';
```

```
import express from "express";
import EventsService from "../services/events-service.js"
import ValidacionesHelper from "../helpers/validaciones-helper.js"
import { authenticateToken } from '../middlewares/auth-middleware.js';
const EventsRouter =  Router();
const svc = new EventsService();

//4
EventsRouter.get('/:id', async (req, res) => {
    try {
```

```
        const  id  = req.params.id;
        const evento = await svc.getById(id);
        if (evento) {
            res.status(200).json(evento);
        } else {
            res.status(404).json({ message: 'Evento no encontrado' });
        }
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

//2
EventsRouter.get('/', async (req, res) => {
    const { name, category, startDate, endDate, page, pageSize } =
req.query;

    try {
        const events = await svc.searchEvents({ name, category,
startDate, endDate, page, pageSize });
        res.status(200).json(events);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

/*EventsRouter.get('/:id', async (req, res) => {
    const id = (req.params.id);
    const result = await svc.getById(id);
    if (result) {
        res.status(200).send(result);
    }
    else {
        res.status(404).send('Evento no encontrado');
    }
});*/

//5 NO FUNCIONA
EventsRouter.get('/:id/enrollment', async (req, res) => {
    const id = (req.params.id);
    if (id === null) {
        res.status(400).send('El id de evento debe ser un número
entero');
        return;
    }
    const firstName = (req.query.first_name);
    const lastName = (req.query.last_name);
    const username = (req.query.username);
    const attended = (req.query.attended);
```

```
    const rating = (req.query.rating);
    const result = await svc.listParticipantes(id, firstName, lastName,
username, attended, rating);
    if (result) {
        res.status(200).send(result);
    }
    else {
        res.status(404).send('No se encontraron inscripciones que cumplan
con los criterios de búsqueda');
    }
});
//8
EventsRouter.post('/', authenticateToken, async (req, res) => {
    const { name, description, max_assistance, max_capacity, price,
duration_in_minutes, id_event_location } = req.body;
    const userId = req.user.id;

    if (!name || !description || name.length < 3 || description.length <
3) {
        return res.status(400).json({ message: 'El nombre o la
descripción son inválidos.' });
    }

    if (max_assistance > max_capacity) {
        return res.status(400).json({ message: 'El max_assistance es
mayor que el max_capacity.' });
    }

    if (price < 0 || duration_in_minutes < 0) {
        return res.status(400).json({ message: 'El precio o la duración
son inválidos.' });
    }

    try {
        const newEvent = await createEvent({ name, description,
max_assistance, max_capacity, price, duration_in_minutes,
id_event_location, userId });
        res.status(201).json(newEvent);
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
});

EventsRouter.put('/', authenticateToken, async (req, res) => {
    const { name, description, max_assistance, max_capacity, price,
duration_in_minutes, id_event_location } = req.body;
    const userId = req.user.id;
    if (!name || !description || name.length < 3 || description.length <
3) {
```

```
        return res.status(400).json({ message: 'El nombre o la
descripción son inválidos.' });
    }

    if (max_assistance > max_capacity) {
        return res.status(400).json({ message: 'El max_assistance es
mayor que el max_capacity.' });
    }

    if (price < 0 || duration_in_minutes < 0) {
        return res.status(400).json({ message: 'El precio o la duración
son inválidos.' });
    }
    if(!userId)// en caso de que no se encuentre autenticado (nose como
verificar eso)
    {
        return res.status(401).json({ message: 'El usuario no se
encuentra autenticado' });
    }
    try {
        const evento = svc.getById(id);
        if (!evento|| evento.user_id !== userId) {
            res.status(404).send('El evento no existe o no le pretenece
al usuario autenticado');
            return;
        }
        else {
            const newEvent = await svc.updateEvent({ name, description,
max_assistance, max_capacity, price, duration_in_minutes,
id_event_location});
            return res.status(200).json(newEvent);
        }
    }catch (error) {
        res.status(500).json({ message: error.message });
    }
});
EventsRouter.delete('/:id', authenticateToken, async (req, res) => {
    const id=req.params.id;
    const userId = req.user.id;
    try {
        const evento = svc.getById(id);
        if (!evento|| evento.user_id !== userId) {
            res.status(404).send('El evento no existe o no le pretenece
al usuario autenticado');
            return;
        }
        if(!userId)// en caso de que no se encuentre autenticado (nose
como verificar eso)
        {
```

```
            return res.status(401).json({ message: 'El usuario no se
encuentra autenticado' });
        }
        const hasUsersRegistered = false; // Valida que no haya usuarios
registrados a este evento ¿?
        if (hasUsersRegistered) {
            return res.status(400).json({ message: 'Hay usuarios
registrados en el evento.' });
        }
        else {
            await svc.deleteEvent(id);
            return res.status(200).json({ message: 'Evento eliminado
correctamente.'});
        }
    }catch (error) {
        res.status(500).json({ message: error.message });
    }
});
//

EventsRouter.post('/:id/enrollment', authenticateToken, async (req, res)
=> {
    const id = req.params.id
    if (id === null) {
        res.status(400).send('El id de evento debe ser un número
entero');
        return;
    }
    const Id = svc.getById(id);
    if (!Id) {
        res.status(404).send('Evento no encontrado');
        return;
    }
    const rest = await svc.enrollAsync(id, req.user.id);
    if (rest) {
        res.status(201).send();
    }
    else {
        res.status(400).send('Ya no hay cupos disponibles');
    }
})

EventsRouter.delete('/:id/enrollment', authenticateToken, async (req,
res) => {
    const id = req.params.id
    if (id === null) {
        res.status(400).send('El id de evento tiene que ser un número
entero');
        return;
```

```
    }
    const rest = await svc.unenrollAsync(id, req.user.id);
    if (rest == 200) {
        res.status(200).send();
    }
    else if (rest == 404) {
        res.status(404).send('Inscripción o evento no encontrado');
    }
    else {
        res.status(400).send('El evento ya pasó');
    }
});

EventsRouter.patch('/:id/enrollment/:rating', authenticateToken, async
(req, res) => {
    const eventId = req.params.id;
    const rating = req.params.rating;
    const userId = req.user.id;
    const { observations } = req.body;
    let respuesta;
    try {
        await rateEvent(eventId, userId, rating, observations);
        respuesta = res.status(200).json({ message: 'Evento rankeado
correctamente.' });
    } catch (error) {
        respuesta = res.status(error.status || 500).json({ message:
error.message });
    }
    return respuesta;
});

export default EventsRouter;
```