

Abigail Gonzales Hidalgo

A00819967

Fernanda Montaña Rios

A01730440

La aplicación es un chatbot con el cual te comunicas cuando vas a salir, se le puede introducir lugares frecuentes para su fácil acceso dentro del chatbot. Aparte de poder seleccionar un lugar, también se selecciona hora y con ella al pasar el tiempo seleccionado el chatbot aguardara para preguntar la llegada del usuario. Su objetivo es mantenerte seguro durante todo tu trayecto.

Las librerías utilizadas fueron:

Front-end

- React Simple Chatbot: Qué fue utilizada para la implementación del chatbot utilizado.
- Material-ui: Fue seleccionada para la implementación del front-end, fue de gran utilidad
- React-router-dom: Manejo de componentes
- Reflux: Manejo de componentes

```
src/components/dashboard/Chat/ChatComponentWithout
import React, { Component, useState } from 'react';
import PropTypes from 'prop-types';
import ChatBot from 'react-simple-chatbot';
import { ThemeProvider } from 'styled-components';
import Grid from '@material-ui/core/Grid';
import Fab from '@material-ui/icons/AccountCircle';
import Account from '@material-ui/icons/AccountCircle';
import { BrowserRouter as Router, Route, Link, } from "react-router-dom";
import Container from '@material-ui/core/Container';
import { positions } from '@material-ui/system';
import Exit from '@material-ui/icons/ExitToApp';
```

Back-end

- Guardian: para la autenticación
- Uberauth: para la autenticación
- CORSPlug: para el manejo de requests
- Comeonin: para la autenticación
- pbkdf2\_elixir: Hasheo de contraseña

```
defp deps do
  [
    {:phoenix, "~> 1.4.10"},
    {:phoenix_pubsub, "~> 1.1"},
    {:phoenix_ecto, "~> 4.0"},
    {:ecto_sql, "~> 3.1"},
    {:postgrex, ">= 0.0.0"},
    {:gettext, "~> 0.11"},
    {:jason, "~> 1.0"},
    {:plug_cowboy, "~> 2.0"},
    {:ueberauth, "~> 0.5.0"},
    {:ueberauth_identity, "~> 0.2"},
    {:guardian, "~> 1.0"},
    {:comeonin, "~> 4.1"},
    {:pbkdf2_elixir, "~> 0.12"},
    {:cors_plug, "~> 2.0"}
  ]
end
```

## Muestra de las tablas desde la Shell de PSQL

```
SQL Shell (psql)
Los caracteres de 8 bits pueden funcionar incorrectamente.
Vea la página de referencia de psql «Notes for Windows users»
para obtener más detalles.
Ahora está conectado a la base de datos «ada_be_dev» con el usuario «postgres».
ada_be_dev=# select * from groups;
 id | name | description | id_group | inserted_at | updated_at
-----+-----+-----+-----+-----+-----
 1 | Casita | donde esta el cora | 666 | 2019-12-08 19:16:17 | 2019-12-08 19:16:17
(1 fila)

ada_be_dev=# select * from user_groups;
 user_id | group_id
-----+-----
 3 | 1
(1 fila)

ada_be_dev=# select * from users;
 id | email | hashed_password | username
-----+-----+-----+-----
 1 | admin@admin.com | $pbkdf2-sha512$160000$vnvsnfF6Lyj3T2VMU8VikQ$GacrfEUNmSPMjW0MQK1a5T18YutCjQ9o5SoQhX2Xmjg0FatInfR8zbGL16YbvqK4wk.ca0c53Ij1/6cHUonSpg | admin
 2019-12-07 23:31:50 | 2019-12-07 23:31:50
 2 | ab1@abi.com | $pbkdf2-sha512$160000$95mp2IuCjRwzPreapPqgg$rnZnZwH7go1IFQj0p24dgiGF8.926R./Y.cj6s9iT6DBi3m9zU3vJh/PBhqjxLE6YtHhWqVE4fSXL1o.5wr/HQAA | abi
 2019-12-07 23:31:50 | 2019-12-07 23:31:50
 3 | fer@fer.com | $pbkdf2-sha512$160000$44FqD8XgfrPm56bZKXrXVTQ$BxN64d3MeumUJk/BVFS2EXopqty.07d4tzQg17BFMaFNYYwBSzazc3FmmucXXYpA.9qHToQmc2zEQX/HDMf3g | fer
 2019-12-07 23:31:50 | 2019-12-07 23:31:50
(3 filas)

ada_be_dev=# select * from places;
 id | name | description | user_id | inserted_at | updated_at
-----+-----+-----+-----+-----+-----
 1 | Tec | escuela | 3 | 2019-12-07 23:33:13 | 2019-12-07 23:33:13
 2 | casita | ya casi :3 | 3 | 2019-12-07 23:33:56 | 2019-12-07 23:33:56
 3 | cine | palomitas | 3 | 2019-12-07 23:34:29 | 2019-12-07 23:34:29
 4 | Biblioteca | libros | 3 | 2019-12-08 19:00:29 | 2019-12-08 19:00:29
 5 | Kiba | donde vive el chema | 3 | 2019-12-09 03:56:39 | 2019-12-09 03:56:39
 6 | dkd | d | 3 | 2019-12-09 04:25:18 | 2019-12-09 04:25:18
 7 | parque | con kayro | 3 | 2019-12-09 16:53:44 | 2019-12-09 16:53:44
(7 filas)
```

## Migrations del backend:

```
defmodule AdaBe.Repo.Migrations.CreateUsers do
  use Ecto.Migration

  def change do
    create table(:users) do
      add :email, :string, null: false
      add :hashed_password, :string, null: false
      add :username, :string, null: false

      timestamps()
    end

    create unique_index(:users, [:email])
  end
end
```

```
defmodule AdaBe.Repo.Migrations.CreatePlaces do
  use Ecto.Migration

  def change do
    create table(:places) do
      add :name, :string, null: false
      add :description, :string, null: false
      add :user_id, references(:users, on_delete: :nothing), null: false

      timestamps()
    end

    create unique_index(:places, [:name])
    create index(:places, [:user_id])
  end
end
```

```
defmodule AdaBe.Repo.Migrations.CreateGroups do
  use Ecto.Migration

  def change do
    create table(:groups) do
      add :name, :string
      add :description, :string
      add :id_group, :integer

      timestamps()
    end

    create unique_index(:groups, [:id_group])
  end
end
```

```
defmodule AdaBe.Repo.Migrations.UserGroups do
  use Ecto.Migration

  def change do
    create table(:user_groups, primary_key: false) do
      add(:user_id, references(:users, on_delete: :delete_all), primary_key: true)
      add(:group_id, references(:groups, on_delete: :delete_all), primary_key: true)
    end

    create(index(:user_groups, [:user_id]))
    create(index(:user_groups, [:group_id]))

    create(unique_index(:user_groups, [:user_id, :group_id]))

  end
end
```

Los schemas de las respectivas migations:

```
defmodule AdaBe.Accounts.User do
  use Ecto.Schema
  import Ecto.Changeset

  schema "users" do
    field :username, :string
    field :password, :string, virtual: true
    field :password_confirmation, :string, virtual: true
    field :email, :string
    field :hashed_password, :string
    has_many(:places, AdaBe.Menu.Place)
    many_to_many(:groups, AdaBe.Menu.Group, join_through: "user_groups", on_replace: :delete)

    timestamps()
  end

  @doc false
  def changeset(user, attrs) do
    user
    |> cast(attrs, [:username, :password, :password_confirmation, :email])
    |> validate_required([:username, :password, :password_confirmation, :email])
    |> validate_format(:email, ~r/@/)
    |> unique_constraint(:email)
    |> validate_length(:password, min: 8)
    |> validate_confirmation(:password)
    |> put_hashed_password()
  end

  defp put_hashed_password(changeset) do
    case changeset do
      %Ecto.Changeset{valid?: true, changes: %{password: password}} ->
        put_change(changeset, :hashed_password, Comeonin.Pbkdf2.hashpwsalt(password))
      _ ->
        changeset
    end
  end
end
```

Con el `authentication_controller.ex` se hace el control de autenticación de las cuentas de los usuarios:

```
defmodule AdaBeWeb.AuthenticationController do
  use AdaBeWeb, :controller

  alias AdaBe.Accounts

  plug Ueberauth

  def identity_callback(%{assigns: %{ueberauth_auth: auth}} = conn, _params) do
    email = auth.uid
    password = auth.credentials.other.password
    handle_user_conn(Accounts.get_user_by_email_and_password(email, password), conn)
  end

  def delete(conn, _params) do
    conn
    |> AdaBeWeb.Guardian.Plug.sign_out()
    |> json(%{message: "logout realizado"})
  end

  defp handle_user_conn(user, conn) do
    case user do
      {:ok, user} ->
        {:ok, jwt, _full_claims} =
          {:ok, jwt, _full_claims} =
            AdaBeWeb.Guardian.encode_and_sign(user, %{})

        conn
        |> put_resp_header("authorization", "Bearer #{jwt}")
        |> json(%{data: %{token: jwt}})

      # Handle our own error to keep it generic
      {:error, _reason} ->
        conn
        |> put_status(401)
        |> json(%{message: "Authentication error"})
    end
  end
end
```

```

defmodule AdaBeWeb.GroupController do
  use AdaBeWeb, :controller
  import Ecto.Query, only: [from: 2]

  alias AdaBe.Menu.Group
  alias AdaBe.Menu.UserGroups
  alias AdaBe.Accounts.User
  alias AdaBe.Repo

  def create(conn, %{"group" => group_params}) do
    user = AdaBeWeb.Guardian.Plug.current_resource(conn)
    user = Repo.preload(user, [:groups])
    group = Group.changeset(%Group{}, group_params)
    groups = user.groups ++ [group] |> Enum.map(&Ecto.Changeset.change/1)
    user
    |> Ecto.Changeset.change
    |> Ecto.Changeset.put_assoc(:groups, groups)
    |> Repo.update

    IO.inspect "====="

    json(conn, %{msg: "Group successfully registered"})
  end
end

```

Con group\_controller.ex se hace el control cuando se crea un grupo

```

defmodule AdaBeWeb.GroupController do
  use AdaBeWeb, :controller
  import Ecto.Query, only: [from: 2]

  alias AdaBe.Menu.Group
  alias AdaBe.Menu.UserGroups
  alias AdaBe.Accounts.User
  alias AdaBe.Repo

  def create(conn, %{"group" => group_params}) do
    user = AdaBeWeb.Guardian.Plug.current_resource(conn)
    user = Repo.preload(user, [:groups])
    group = Group.changeset(%Group{}, group_params)
    groups = user.groups ++ [group] |> Enum.map(&Ecto.Changeset.change/1)
    user
    |> Ecto.Changeset.change
    |> Ecto.Changeset.put_assoc(:groups, groups)
    |> Repo.update

    IO.inspect "====="

    json(conn, %{msg: "Group successfully registered"})
  end
end

```

```
defmodule AdaBeWeb.JoinController do
  use AdaBeWeb, :controller

  alias AdaBe.Menu.Group
  alias AdaBe.Menu.UserGroups
  alias AdaBe.Accounts.User
  alias AdaBe.Repo

  def join(conn, %{"group" => group_params}) do
    user = AdaBeWeb.Guardian.Plug.current_resource(conn)
    user = Repo.preload(user, [:groups])
    group_id = String.to_integer(Map.fetch!(group_params, "id_group"))
    group = Repo.get_by(Group, id_group: group_id)
    groups = user.groups ++ [group] |> Enum.map(&Ecto.Changeset.change/1)
    user
    |> Ecto.Changeset.change
    |> Ecto.Changeset.put_assoc(:groups, groups)
    |> Repo.update
    IO.inspect "======"
    json(conn, %{msg: "You have been successfully registered"})
  end
end
```

Con join\_controller.ex se hace el control cuando un usuario se une a un grupo con el identificador numérico especial de dicho grupo.

```
defmodule AdaBeWeb.PlaceController do
  use AdaBeWeb, :controller
  import Ecto.Query, only: [from: 2]

  alias AdaBe.Menu.Place
  alias AdaBe.Accounts.User
  alias AdaBe.Repo

  def index(conn, _params) do
    user = AdaBeWeb.Guardian.Plug.current_resource(conn)

    names = Repo.all(from p in Place, where: p.user_id == ^user.id, select: p.name)

    json(conn, %{msg: "Hello", names: names})
  end

  def create(conn, %{ "place" => place_params }) do
    user = AdaBeWeb.Guardian.Plug.current_resource(conn)
    Place.changeset(%Place{}, place_params)
    |> Ecto.Changeset.put_change(:user_id, user.id)
    |> IO.inspect
    |> Repo.insert!()
    IO.inspect "======"

    json(conn, %{msg: "Place successfully registered"})
  end
end
```

Con place\_controller.ex se controla la creación de lugares y el muestreo de estos para el frontend.

```
defmodule AdaBeWeb.RegistrationController do
  use AdaBeWeb, :controller
  alias AdaBe.Accounts.User
  alias AdaBe.Repo

  def register(conn, params) do
    IO.inspect params
    {:ok, _user } = User.changeset(%User{}, params) |> Repo.insert()
    conn |> json( %{msg: "User was successfully registered"} )
  end
end
```

Con el registration\_controller.ex se lleva el control del registro de los usuarios.



Abigail Gonzales Hidalgo

A00819967

Fernanda Montaña Rios

A01730440

```
defmodule AdaBeWeb.GroupChannel do
  use AdaBeWeb, :channel

  def join("groups:" <> id_group, _params, socket) do
    group = Repo.get!(AdaBe.Menu.Group, id_group)

    {:ok, assign(socket, :group, group)}
  end

  def terminate(_reason, socket) do
    {:ok, socket}
  end
end
```

Tenemos el group\_channel.ex que une a los usuarios al channel del grupo al que estén unidos para recibir las notificaciones de sus contactos de los respectivos grupos.

```

defmodule AdaBeWeb.UserSocket do
  use Phoenix.Socket

  ## Channels
  channel "room:*", AdaBeWeb.RoomChannel

  # Socket params are passed from the client and can
  # be used to verify and authenticate a user. After
  # verification, you can put default assigns into
  # the socket that will be set for all channels, ie
  #
  #   {:ok, assign(socket, :user_id, verified_user_id)}
  #
  # To deny connection, return `:error`.
  #
  # See `Phoenix.Token` documentation for examples in
  # performing token verification on connect.
  def connect(%{"token" => token}, socket) do
    case Guardian.decode_and_verify(token) do
      {:ok, claims} ->
        case AdaBeWeb.GuardianSerializer.from_token(claims["sub"]) do
          {:ok, user} ->
            {:ok, assign(socket, :current_user, user)}
          {:error, _reason} ->
            :error
        end
      {:error, _reason} ->
        :error
    end
  end

  def connect(_params, _socket), do: :error
  # Socket id's are topics that allow you to identify all sockets for a given user:
  #
  # def id(socket), do: "user_socket:#{socket.assigns.user_id}"
  #
  # Would allow you to broadcast a "disconnect" event and terminate
  # all active sockets and channels for a given user:
  #
  #   AdaBeWeb.Endpoint.broadcast("user_socket:#{user.id}", "disconnect", %{})

```

User\_socket.ex controla el manejo de canales.

```
import React, {useState} from 'react';
import './App.css';
import { BrowserRouter as Router, Route } from "react-router-dom";

import Landing from "../components/layout/Landing";
import Chat from "../components/dashboard/Chat";
import Login from "../components/users/Login";
import Register from "../components/users/Register";
import Profile from "../components/dashboard/Profile";

function App() {
  const [token, setToken] = useState("");

  return (
    <Router>
      <Route exact strict path="/" component={Landing} />
      <Route exact path="/chat" component={() => <Chat token={token} />} />
      <Route exact path="/register" component={Register} />
      <Route exact path="/login" component={() => <Login setToken={setToken} />} />
      <Route exact path="/profile" component={() => <Profile token={token}/>} />
    </Router>
  );
}

export default App;
```

El app.js controla los diferentes componentes de las paginas

```
let { from } = location.state || { from: { pathname: "/" } };
console.log(props);
let handleAuthentication = async () => {
  fetch("http://localhost:4000/api/auth/identity/callback", {
    method: "POST",
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      user: {
        email: email,
        password: password
      }
    })
  })
  .then( response => {
    if (!response.ok){
      throw alert("Favor de ingresar datos validos")
    }
    return response.json();
  }).then( json => {
    console.log( json.data );
    Actions.login(email, json.data.token );
    props.setToken(json.data.token);
    history.replace("/chat");
  }).catch(error => console.log(error));
}
```

El login.js maneja la autenticación de usuario para el inicio de sesión

```

let { from } = location.state || { from: { pathname: "/" } };
console.log(from);
let registration = async () => {
  fetch("http://localhost:4000/api/registration", {
    method: "POST",
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      username: username,
      email: email,
      password: password,
      password_confirmation: password_confirmation
    })
  })
  .then( response => {
    if (!response.ok)
      throw new Error("error")
    return response.json();
  }).then( json => {
    console.log( json.data );
    history.replace("/login");
  })
}

```

El Register.js maneja el registro de usuarios para su futuro inicio de sesión

```

<Grid container style={{height:"100vh"}}>
  <Grid textAlign="center" width="50%" justify="flex-start" item xs={12} lg={6} style={{backgroundColor: '#FFE0EB', color: 'black'}}>
    <div class="centered" style={{display:'flex'}}>
      <div style={{width:'80%'}}>
        <ThemeProvider theme={theme}>
          {
            this.state.loaded ? (<ChatBot headerTitle="Ada"
              speechSynthesis={{ enable: true, lang: 'sp' }}
              steps=[
                {
                  id: '1',
                  message: 'A que lugar vas?',
                  trigger: 'lugar'
                },
                {
                  id: 'lugar',
                  options: this.state.options
                },
                {
                  id: '3',
                  message: 'Cuanto tiempo vas a tardar?',
                  trigger: 'tiempo',
                },
                {
                  id: 'tiempo',
                  options: [
                    { value: '60000', label: '1 min', trigger: '7' },
                    { value: '900000', label: '15 min', trigger: '7' },
                    { value: '1800000', label: '30 min', trigger: '7' },
                    { value: '3600000', label: '1 hora', trigger: '7' },
                    { value: '7200000', label: '2 horas', trigger: '7' },
                  ],
                },
              ],
            ) : null
          }
        </ThemeProvider>
      </div>
    </div>
  </Grid>
</Grid>

```

```
import React, { Component, useState } from 'react';
import PropTypes from 'prop-types';
import ChatBot from 'react-simple-chatbot';
import { ThemeProvider } from 'styled-components';
import Grid from '@material-ui/core/Grid';
import Fab from '@material-ui/core/Fab';
import Account from '@material-ui/icons/AccountCircle';
import { BrowserRouter as Router, Route, Link, } from "react-router-dom";
import Container from '@material-ui/core/Container';
import { positions } from '@material-ui/system';
import Exit from '@material-ui/icons/ExitToApp';

class Review extends Component {
  constructor(props) {
    super(props);
    this.state = {
      lugar: '',
      tiempo: '',
    };
  }

  componentWillMount() {
    const { steps } = this.props;
    const { lugar, tiempo } = steps;
    this.setState({ lugar, tiempo });
  }

  render() {
    const { lugar, tiempo } = this.state;
    return (
      <div style={{ width: '100%' }}>
        <h3>informacion</h3>
        <table>
          <tbody>
            <tr>
              <td>Lugar:</td>
              <td>{lugar.value}</td>
            </tr>
            <tr>
              <td>Tiempo:</td>
            </tr>
          </tbody>
        </table>
      </div>
    );
  }
}
```

El chat.js controla todo lo relacionado con el chatbot, así como la información que consigue.

```
export default function Profile(props) {
  const classes = useStyles();

  console.log(`El token es ${props.token}`)
  let history = useHistory();
  let location = useLocation();
  const [name, setName] = useState("");
  const [description, setDescription] = useState("");
  const [name_g, setNameg] = useState("");
  const [description_g, setDescriptiong] = useState("");
  const [id_g, setId] = useState("");
  const [id, setIdg] = useState("");

  let { from } = location.state || { from: { pathname: "/" } };
  console.log(from);
  let places = async () => {
    fetch("http://localhost:4000/api/places", {
      method: "POST",
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${props.token}`
      },
      body: JSON.stringify({
        place: {
          name: name,
          description: description
        }
      })
    })
  })
  .then( response => {
    if (!response.ok)
      throw alert("Error al crear un lugar")
    return response.json();
  }).then( json => {
    console.log( json.data );
  }).catch(error => console.log(error));
  document.location.href = "/profile";
}
```

En profile.js se controla el ingreso de lugares frecuentes que posteriormente se muestran en el chatbot

```
let groups = async () => {
  fetch("http://localhost:4000/api/groups", {
    method: "POST",
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${props.token}`
    },
    body: JSON.stringify({
      group: {
        name: name_g,
        description: description_g,
        id_group: id_g
      }
    })
  })
  .then( response => {
    if (!response.ok)
      throw new Error("error")
    return response.json();
  }).then( json => {
    console.log( json.data );
  }).catch(error => console.log(error));
  document.location.href = "/profile";
}

let join = async () => {
  fetch("http://localhost:4000/api/join", {
    method: "POST",
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${props.token}`
    },
    body: JSON.stringify({
      group: {
        id_group: id
      }
    })
  })
  .then( response => {
    if (!response.ok)
```

El profile.js, controla también el poder unirse a un grupo y crearlo si el usuario así lo desea

## Conclusión

El proyecto fue muy interesante de realizar ya que se interactuó con un nuevo lenguaje para el backend y nos ayudó para mejorar nuestros conocimientos en elixir. Lo que se complicó fue el manejo de información dentro de la aplicación, ya que el chatbot tenía problemas para el manejo fuera de esta en el momento de pasarla.