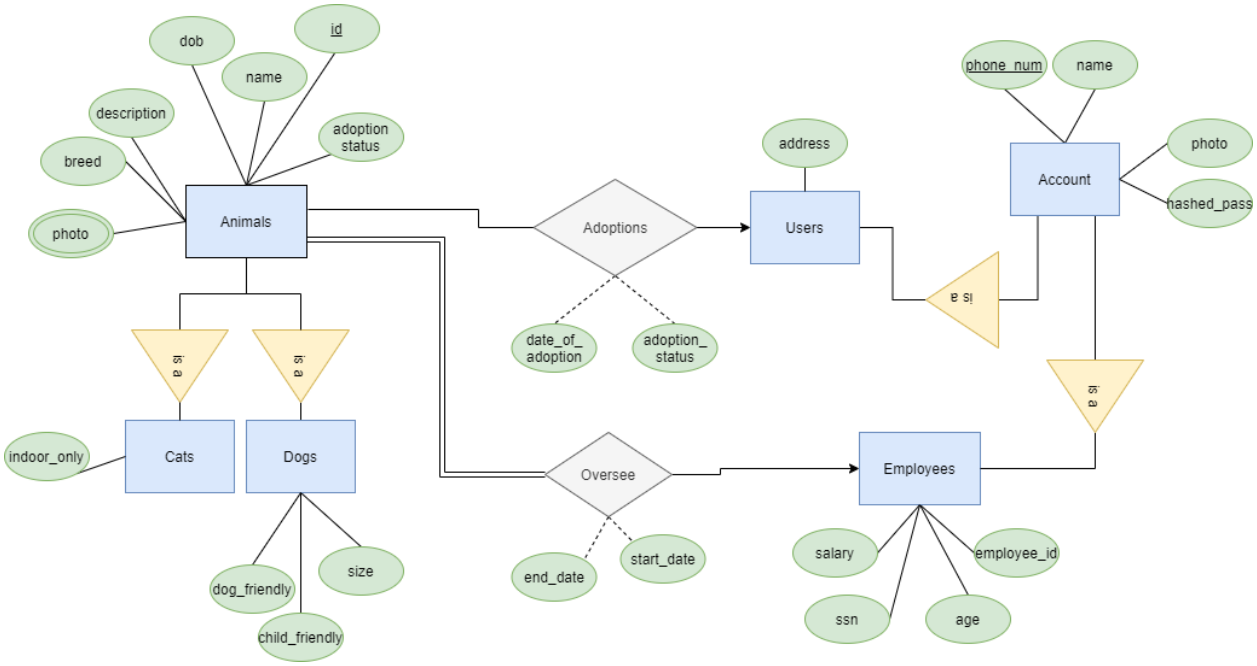
Final Project Group 36: SPCA

Eleanor Ozer, Thai Nguyen, John Nguyen, Edward Kim

Part 1: Database design

https://drive.google.com/file/d/1ZtXiHHtoVDQ_xPOkzh2RSt-pWy733nF0/view?usp=sharing



Convert E-R diagram into tables

- **Animals**(id, name, dob, animaltype, breed, adoption_status, description)
- **Cats**(id, outside_only)
- **Dogs**(id, size, child_friendly, dog_friendly)
- **Animals_photo**(animal_id, photo_id)
- **Accounts**(phone_num, name, photo, hashedpass)

- **Users**(phone_num, address)
- **Employees**(phone_num, employee_id, age, ssn, salary)
- **Adopted**(animal_id, phone, date_of_adoption, adoption_status)
- **Oversee**(employee_id, animal_id, start_date, end_date)

Part 2: Database Programming

We plan on hosting the database locally on XAMPP.
We are planning on using javascript, html, and the react libraries to run our application.
In order to run the project follow these steps:

1) Download the project at https://github.com/eleanorozer/databases-4750
2) Download the necessary dependencies if you're missing any (MUI, node, react, react-scripts,etc.)
3) Import the SQL tables into XAMPP
4) In terminal use "cd spca-app", then use "cd client"
5) In terminal use "npm start"

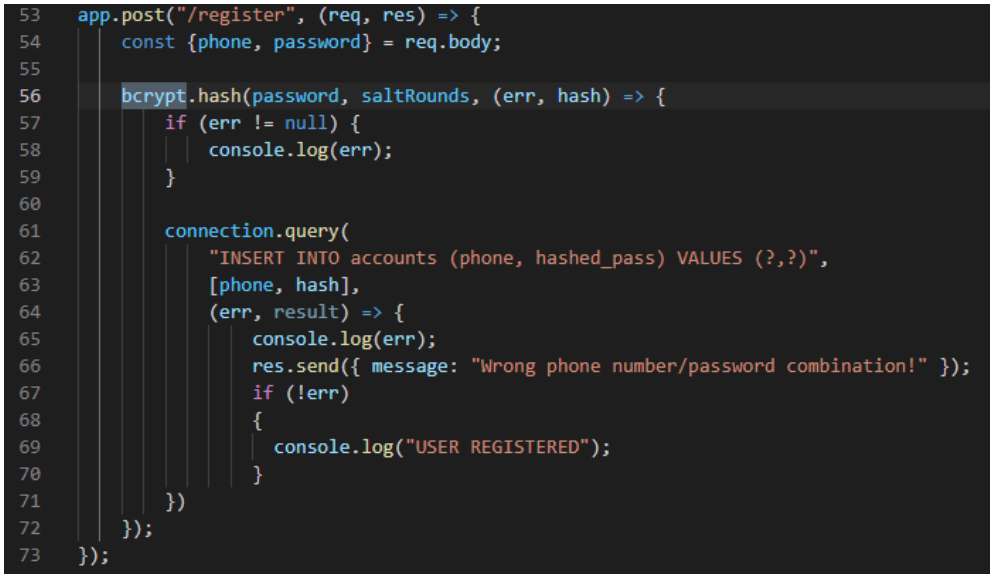
Part 3: Database Security (DB level)

Security is set for the developers in order to ensure that tables in the database can not be directly violated by deleting rows or tables. In addition, we would have security for developers since they would have more permissions compared to regular users. However, we would still have some security for users in the form of password hashing and using third party authentication (just not at the database level but instead in the application level).
Our database would use RBAC (Role-Based Access Control) since admins will have additional powers to add and drop animals in the database while regular users would not have any access.
Here are the admin SQL privileges:
CREATE ROLE admin;
GRANT EXECUTE ON ALL PRIVILEGES TO admin ;
GRANT admin ON users TO 5718391119
GRANT admin ON users TO 5711142216
GRANT admin ON users TO 7031891200
GRANT admin ON users TO 7032812992

Part 4: Database Security (application level)

For our application, we incorporated bcrypt in order to hash user passwords to ensure security as well as using a third party authentication called auth0 in order to handle security in our application.

Below you will find that in our application's **server.js** file we used password hashing with bcrypt based on user input during registration.



```
53  app.post("/register", (req, res) => {
54      const {phone, password} = req.body;
55
56      bcrypt.hash(password, saltRounds, (err, hash) => {
57          if (err != null) {
58              console.log(err);
59          }
60
61          connection.query(
62              "INSERT INTO accounts (phone, hashed_pass) VALUES (?,?)",
63              [phone, hash],
64              (err, result) => {
65                  console.log(err);
66                  res.send({ message: "Wrong phone number/password combination!" });
67                  if (!err)
68                  {
69                    console.log("USER REGISTERED");
70                  }
71          })
72      });
73  });
```

We use password hashing with bcrypt in order to ensure that the passwords are securely stored in the database. Assuming no error was thrown the hashed password would be stored in the database when adding the new user.

Another instance of using bcrypt for hashing is during the login process in **server.js** as well. Below you can see that we compare the imputed hashed password with all passwords with the same phone number since that is the primary key for accounts.

```
83
84  // Login
85  app.post("/login", (req, res) => {
86    // res.json("login");
87    const {phone, password} = req.body;
88
89    connection.query(
90      "SELECT hashed_pass, first_name FROM accounts WHERE phone = ?;",
91      phone,
92      (err, result) => {
93        if (err) {
94          res.send({ err: err });
95        }
96
97        if (result.length > 0) {
98          bcrypt.compare(password, result[0].hashed_pass, (error, response) => {
99            if (response) {
100             req.session.user = result;
101             console.log(req.session.user);
102             res.send(result);
103           } else {
104             res.send({ message: "Wrong phone number/password combination!" });
105           }
106         });
107       } else {
108         res.send({ message: "User doesn't exist" });
109       }
110     }
111   );
112 });
```

Bcrypt ensures that if the password that is tied to that account's phone number is the same as the inputted hashed password it will allow the user to login. Hashed passwords through bcrypt for login and registration helps to ensure that security is implemented at the application level.

Below is the code snippets for auth0.

```
1   import React from 'react';
2   import { useAuth0 } from '@auth0/auth0-react';
3
4   const LoginButton = () => {
5     const { loginWithRedirect } = useAuth0();
6     return (
7       <button
8         className="btn btn-primary btn-block"
9         style={{width: "300px"}}
10        onClick={() => loginWithRedirect({ redirectUri: "http://localhost:3000/home"})}
11        // onClick={() => loginWithRedirect()}
12      >
13        Log In
14      </button>
15    );
16  };
17
18  export default LoginButton;
```

```jsx
import React from 'react';
import { useAuth0 } from '@auth0/auth0-react';
import { button } from 'react-validation/build/button';

const LogoutButton = () => {
  const { logout } = useAuth0();
  return (
    <button
      className="btn btn-danger btn-block"
      style={{width: "100px"}}
      onClick={() =>
        logout({
          returnTo: window.location.origin,
        })
      }
    >
      Log Out
    </button>
  );
};
```

(keep scrolling)

```jsx
import React from 'react';
import { useNavigate } from 'react-router-dom';
import { Auth0Provider } from '@auth0/auth0-react';

const Auth0ProviderWithHistory = ({ children }) => {
  const domain = process.env.REACT_APP_AUTH0_DOMAIN;
  const clientId = process.env.REACT_APP_AUTH0_CLIENT_ID;

  const history = useNavigate();

  const onRedirectCallback = (appState) => {
    history.push(appState?.returnTo || window.location.pathname);
  };

//   const onRedirectCallback = () => {
//     history.push("localhost:3000/profile1");
//   };

  return (
    <Auth0Provider
      domain={domain}
      clientId={clientId}
      redirectUri={window.location.origin}
      onRedirectCallback={onRedirectCallback}
    >
      {children}
    </Auth0Provider>
  );
};

export default Auth0ProviderWithHistory;
```

Auth0 allows us to safely allow users in and out while handling the authentication of the user based on a third party software. This helps ensure that the software side of the application has better security since we are using a trusted third party with regards to security. Auth0 essentially gave us an easy way to implement a secure way to login and logout by redirecting the user to the third party app in all screenshots above.