SOURCE DOCUMENTATION

Below you can find the images of each of the functions used in the source of the project.

1. Connection to Database:

```
const mongoose = require("mongoose");
//link from mongodb atlas to connect to the database
mongoose
  .connect(
    "mongodb+srv://tejasandeep:Phoenix030602@cluster0.ymm5l.mongodb.net/
?retryWrites=true&w=majority"
  )
  .then(() => {
    console.log("Mongoose connection succesful");
  })
  .catch((e) => {
    console.log(e);
  });

//Schema for the database in which the data is stored
const LoginSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true,
  },
  lastName: {
    type: String,
    required: true,
  },
  gender: {
    type: String,
    required: true,
  },
  securityQuestion: {
    type: String,
    required: true,
  },
  answer: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
  password: {
    type: String,
```

```
    required: true,
  },
});
//creating a collection in the database with the name Collection1
const collection = new mongoose.model("Collection1", LoginSchema);

//exporting the collection to be used in other files
module.exports = collection;
```

2. Functional code documentation

```
const express = require("express");
const app = express();
const path = require("path");
const collection = require("./connection");
const alert = require("alert");
//to encrypt password
const bcrypt = require("bcrypt");
// to create a jwt token
const jwt = require("jsonwebtoken");
// to keep secret key in .env file and use it
require("dotenv").config();

const templatePath = path.join(__dirname, "../templates");

app.use(express.json());
// This is for the view engine to load the pages in hbs format
app.set("view engine", "hbs");
app.set("views", templatePath);
app.use(express.urlencoded({ extended: false }));
// This is for the static files like css, js, images
app.use(express.static("templates"));

// This is for the authentication token
let tokens = "";
// This is to save the account details during forgot password usecase
let accountDetails = "";

// To render the home page if the jwt token is valid which is verified
using middle ware
app.get("/", authenticateToken, (req, res) => {
  res.render("home");
  console.log("Logged in successfully");
});

//it loads the signup page
app.get("/signup", (req, res) => {
  res.render("signup");
```

```
});

// It loads the login page
app.get("/login", (req, res) => {
  res.render("login");
});

// It loads the security check page to check for the security question
and answer
app.get("/security-check", (req, res) => {
  res.render("security-check");
});

// It loads the find account page to find the account using email
app.get("/find-account", (req, res) => {
  res.render("find-account");
});

// It loads the reset password page to reset the password
app.get("/reset-password", (req, res) => {
  res.render("reset-password");
});

//It a post request to create a new user and creates one only if the
email do not exist previosly
app.post("/signup", async (req, res) => {
  const check = await collection.findOne({ email: req.body.email });

  if (!check) {
    bcrypt
      .hash(req.body.password, Number(process.env.SALT_ROUNDS))
      .then((hash) => {
        const data = {
          firstName: req.body.firstName,
          lastName: req.body.lastName,
          gender: req.body.gender,
          securityQuestion: req.body.securityQuestion,
          answer: req.body.answer,
          email: req.body.email,
          password: hash,
        };
        collection.insertMany([data]);
        alert("user created");
        res.redirect("/login");
      })
      .catch((e) => {
        console.log(e);
      });
```

```
  } else {
    res.redirect("/login");
    alert("user already exists");
  }
});

// It is a post request to login to the application and creates a jwt
token which expires in 24 hours
app.post("/login", async (req, res) => {
  const check = await collection.findOne({ email: req.body.email });
  if (check) {
    bcrypt
      .compare(req.body.password, check.password)
      .then((result) => {
        if (result) {
          const payload = {
            email: req.body.email,
          };
          const accessToken = jwt.sign(payload, process.env.JWT_KEY, {
            expiresIn: "24h",
          });
          tokens = accessToken;
          res.redirect("/");
        } else {
          res.redirect("/login");
          alert("Incorrect password");
        }
      })
      .catch((e) => {
        console.log(e);
        res.redirect("/login");
        alert("An error occured, Try again");
      });
  } else {
    res.redirect("/login");
    alert("Enter a valid email");
  }
});

// It is a post request to find the account using email along with
checks
app.post("/find-account", async (req, res) => {
  accountDetails = await collection.findOne({ email: req.body.email });
  if (!accountDetails) {
    res.redirect("/find-account");
    alert("Enter an existing email");
  } else {
    res.redirect("/security-check");
```

```
  }
});

// It is a post request to check the security question and answer along
with checks
app.post("/security-check", async (req, res) => {
  if (accountDetails.securityQuestion !== req.body.securityQuestion) {
    res.redirect("/security-check");
    alert("Please select the correct security question");
  } else if (accountDetails.answer !== req.body.answer) {
    res.redirect("/security-check");
    alert("Please enter the correct answer");
  } else if (
    accountDetails.answer === req.body.answer &&
    accountDetails.securityQuestion === req.body.securityQuestion
  ) {
    res.redirect("/reset-password");
  }
});

// It is a post request to reset the password along with checks
app.post("/reset-password", async (req, res) => {
  if (req.body.password !== req.body.confirmPassword) {
    res.redirect("/reset-password");
    alert("Password and Confirm Password should be same");
  } else {
    bcrypt
      .hash(req.body.password, Number(process.env.SALT_ROUNDS))
      .then(async (hash) => {
        await collection.findByIdAndUpdate(accountDetails.id, {
          $set: { password: hash },
        });
      });
  }

  res.redirect("/login");
  alert("Password changed successfully");
});

// It is a get request to logout of the application also clears the jwt
token
app.get("/logout", async (req, res) => {
  try {
    console.log("logout Successfully");
    tokens = "";
    res.redirect("/login");
  } catch (err) {
    res.status(500).send(err);
```

```
  }
});

// This is the middleware to authenticate the jwt token
function authenticateToken(req, res, next) {
  const token = tokens;

  if (token === null) return res.sendStatus(401);

  jwt.verify(token, process.env.JWT_KEY, (err, payload) => {
    if (err) {
      return res.redirect("/login");
    }
    req.email = payload.email;
    next();
  });
}

// This is to listen to the port 3000
app.listen(3000, () => {
  console.log("Server connected listening to port 3000");
});
```