Repository Code

app.js

```
const express = require("express");
const routes = require("./routes");
const cors = require("cors");

const db = require("./models");

const app = express();

app.use(express.json());
app.use(cors());

routes(app);

async function createDBConnection() {
  try {
    await db.sequelize.sync({ force: false});
    console.log("Connection has been established successfully.");
  } catch (error) {
    console.error("Unable to connect to the database:", error);
  }
}

createDBConnection();
app.listen(5000, () => console.log("Server is running on port 5000"));
```

config/index.js

```
const path = require("path");
require("dotenv").config({ path: path.join(__dirname, "../.env") });

module.exports = {
  host: process.env.host || "localhost",
  user: process.env.user || "postgres",
  password: process.env.password || "postgres",
  database: process.env.database || "intQuo",
  dialect: process.env.dialect || "postgres",
  port: process.env.port || "5432",
  secret: process.env.secret || "jayshreeram",
};
```

constants/index.js

```
const constants = {
    user_not_found : "User not found",
    user_already_exists : "User already exists",
    user_created : "User created successfully",
    user_deleted : "User deleted successfully",
    user_updated : "User updated successfully",

    message : "Hello World"
}
```

```
module.exports = constants;
```

controllers/auth/index.js

```
const { getUserService } = require("../../service/user/index");
const { compare } = require("../../utils/encrypt");
const JsonWebToken = require("../../utils/jwt");
const config = require("../../config");

const login = async (req, res) => {
  const { email, password } = req.body;

  const user = await getUserService.byEmail(email);
  const user_password = user.password;

  const valid = await compare(password, user_password);
  console.log(valid);
  if (valid) {
    const jwtUtil = new JsonWebToken(config.secret);
    const accessToken = jwtUtil.generate({ id: user.id }, 10000000);
    const refreshToken = jwtUtil.generate({ id: user.id }, 10000000);

    delete user.dataValues.password;

    return res
      .status(200)
      .json({ ...user.dataValues, accessToken, refreshToken });
  }
  return res.status(200).json({ message: "Incorrect Password" });
};

module.exports = { login };
```

controllers/root/index.js

```
const constants = require("../../constants");

const { message } = constants;

const getRoot = (req, res) => {
  res.status(200).send(message);
};

module.exports = { getRoot };
```

controllers/users/index.js

```
const JsonWebToken = require("../../utils/jwt");

const {
  getUserService,
  postUserService,
  putUserService,
  deleteUserService,
} = require("../../service/user/index");

const constants = require("../../constants");

const config = require("../../config");

const { encrypt } = require("../../utils/encrypt");

const {
  user_not_found,
  user_already_exists,
  user_created,
  user_deleted,
  user_updated,
} = constants;

const getAllUsers = async (req, res) => {
  try {
    const users = await getUserService.all();
    users.map((user) => delete user.dataValues.password);
    res.status(200).json({ data: users });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

const getUserById = async (req, res) => {
  try {
    const user = await getUserService.byId(req.params.id);
    console.log(user);
    if (!user[0]) return res.status(404).json({ message: user_not_found });
    delete user[0].dataValues.password;
    res.status(200).json({ data: user[0] });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

const getMe = async (req, res) => {
  try {
    return res.status(200).json(req.user);
  } catch (e) {
    return res.status(500).send(e);
  }
};

const postUser = async (req, res) => {
  try {
    const userExists = await getUserService.byEmail(req.body.email);
    console.log(userExists);

    if (!userExists) {
      const userData = {
        ...req.body,
        password: await encrypt(req.body.password),
      };
      const user = await postUserService.create(userData);

      const jwtUtil = new JsonWebToken(config.secret);
      const accessToken = jwtUtil.generate({ id: user.id }, 10000000);
      const refreshToken = jwtUtil.generate({ id: user.id }, 10000000);

      delete user.dataValues.password;

      res
        .status(201)
        .json({
          data: { ...user.dataValues, accessToken, refreshToken },
          message: user_created,
        });
    } else {
      res.status(400).json({ message: user_already_exists });
    }
  } catch (error) {
    console.log(error);
    res.status(500).json({ error: error.message });
  }
};

const putUser = async (req, res) => {
  try {
    const user = await putUserService.byId(req.params.id, req.body);

    if (!user[0]) return res.status(404).json({ message: user_not_found });

    res.status(200).json({ message: user_updated });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

const deleteUser = async (req, res) => {
  try {
    const del = await deleteUserService.byId(req.params.id);

    if (!del) return res.status(404).json({ message: user_not_found });
    else return res.status(200).json({ message: user_deleted });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

module.exports = {
  getAllUsers,
  getUserById,
  getMe,
  postUser,
  putUser,
  deleteUser,
};
```

initializer/db.js

```
const Sequelize = require("sequelize");
const config = require("../config");

console.log(config);

const sequelize = new Sequelize(config.database, config.user, config.password, {
  host: config.host,
  dialect: config.dialect,
  port: config.port,
  logging: console.log,
});

module.exports = sequelize;
```

middleware/auth.js

```
const JsonWebToken = require("../utils/jwt");
const { getUserService } = require("../service/user/index");
const config = require("../config/index");

const auth = async (req, res, next) => {
  let token;
  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith("Bearer")
  ) {
    try {
      token = req.headers.authorization.split(" ")[1];

      const jwtUtil = new JsonWebToken(config.secret);
      const decoded = await jwtUtil.decode(token);

      if (!decoded) {
        return res.status(401).json({ message: "Unauthorised request" });
      }

      req.user = await getUserService.byId(decoded.id);

      next();
    } catch (error) {
      res.status(401).send(error);
    }
  }
  if (!token) {
    res.status(401);
    throw new Error("Not authorized, no token");
  }
};

module.exports = auth;
```

models/index.js

```
const user = require("./user/index.js");
const interview = require("./interview/index.js");
const question = require("./question/index.js");

const sequelize = require("../initializer/db.js");
const { DataTypes } = require("sequelize");

const db = {};

function connectModels() {
  db.User = user(sequelize, DataTypes);
  db.Interview = interview(sequelize, DataTypes);
  db.Question = question(sequelize, DataTypes);
  associateModels();
}

function associateModels() {
  db.User.associate(db);
  db.Interview.associate(db);
  db.Question.associate(db);
}
connectModels();
db.sequelize = sequelize;

module.exports = db;
```

models/interview/index.js

```
                                        module.exports = (sequelize, DataTypes) => {
  const Interview = sequelize.define(
    "Interview",
    {
      id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
      },
      job_role: {
        type: DataTypes.STRING,
        allowNull: true,
      },
      compensation: {
        type: DataTypes.INTEGER,
        allowNull: true,
      },
      conducted_on: {
        type: DataTypes.DATE,
      },
      status: {
        type: DataTypes.ENUM("On-campus", "Off-campus"),
        allowNull: false,
      },
      result: {
        type: DataTypes.ENUM("Selected", "Rejected"),
      },
    },
    {
      paranoid: true,
    }
  );

  Interview.associate = (models) => {
    Interview.belongsTo(models.User, {
      foreignKey: {
        name: "candidate_id",
        allowNull: true,
      },
    });
  };

  return Interview;
};
```

models/question/index.js

```
module.exports = (sequelize, DataTypes) => {
  const Question = sequelize.define(
    "Question",
    {
      id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
      },
      question: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      user_answer: {
        type: DataTypes.STRING,
        allowNull: true,
      },
      ai_answer: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      difficulty: {
        type: DataTypes.ENUM("Easy", "Medium", "Hard"),
        allowNull: false,
      },
      topic: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      subtopic: {
        type: DataTypes.STRING,
        allowNull: false,
      },
    },
    {
      paranoid: true,
    }
  );
```

```
    Question.associate = (models) => {
        Question.belongsTo(models.Interview, {
            foreignKey: {
                name: "interview_id",
                allowNull: false,
            },
        });
    }

  return Question;
};
```

models/user/index.js

```
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define(
    "User",
    {
      id: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
      },
      username: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      first_name: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      last_name: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      email: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      passout_year: {
        type: DataTypes.INTEGER,
        allowNull: false,
      },
      password: {
        type: DataTypes.STRING,
        allowNull: false,
      },
    },
    {
      paranoid: true,
    }
  );

  User.associate = (models) => {
    User.hasMany(models.Interview, {
      foreignKey: {
        name: "candidate_id",
        allowNull: false,
      },
    });
  };

  return User;
};
```

package.json

```
{
  "name": "node-express-template",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.2",
    "pg": "^8.11.3",
    "sequelize": "^6.35.0"
  },
  "devDependencies": {
    "babel-cli": "^6.26.0"
  }
}
```

routes/auth/index.js

```
const express = require("express");

const auth_controller = require("../../controllers/auth/index");

const auth_router = express.Router();

auth_router.post("/login", auth_controller.login);

module.exports = auth_router;
```

routes/index.js

```
const root = require("./root/index.js");
const user = require("./user/index.js");
const auth = require("./auth/index.js");

const routes = (app) => {
  app.use("/", root);
  app.use("/auth", auth);
  app.use("/user", user);
};

module.exports = routes;
```

routes/root/index.js

```
const express=require("express")
const root_controller=require("../../controllers/root/index.js")


const root=express.Router()

root.get("/",root_controller.getRoot)

module.exports=root;
```

routes/user/index.js

```
const router = require("express").Router();
const userController = require("../../controllers/users/index");
const auth = require("../../middleware/auth");
router.get("/", userController.getAllUsers);
router.get("/me", auth, userController.getMe);
router.get("/:id", userController.getUserById);
router.post("/", userController.postUser);
router.put("/:id", userController.putUser);
router.delete("/:id", userController.deleteUser);

module.exports = router;
```

service/index.js

service/user/index.js

```
const getUserService = require("./userServices/get");
const postUserService = require("./userServices/post");
const putUserService = require("./userServices/put");
const deleteUserService = require("./userServices/delete");

module.exports = {
  getUserService,
  postUserService,
  putUserService,
  deleteUserService,
};
```

service/user/userServices/delete.js

```
const db=require("../../../models/index");

const byId=async(id)=>{
    return await db.User.destroy({where:{id}});
};

module.exports={
    byId
};
```

service/user/userServices/get.js

```
const db = require("../../../models/index");

const all = async () => {
    console.log(db.User);
  return await db.User.findAll({ include: { all: true } });
};

const byId = async (id) => {
  return await db.User.findAll({ where: { id } });
};

const byEmail = async (email) => {
  return await db.User.findOne({ where: { email } });
};

module.exports = {
  all,
  byId,
  byEmail,
};
```

service/user/userServices/post.js

```
const db = require("../../../models/index");

const create = async (user) => {
  return await db.User.create(user);
};

module.exports = {
  create,
};
```

service/user/userServices/put.js

```
const db = require("../../../models/index");

const byId = async (id, user) => {
  return await db.User.update(user, { where: { id } });
};

module.exports = {
  byId,
};
```

utils/encrypt.js

```
const bcrypt = require('bcryptjs');

const encrypt = async (password) => {
    const pass=await bcrypt.hash(password, 10);
    console.log(pass);
    return pass;
}
```

```
const compare = async (password, hash) => {
    return await bcrypt.compare(password, hash);
}

module.exports = {
    encrypt,
    compare
}
```

utils/jwt.js

```
const jsonwebtoken = require("jsonwebtoken");

class JsonWebToken {
    #serverSecret;

    constructor(serverSecret) {
        this.serverSecret = serverSecret;
    }

    decode(token) {
        return jsonwebtoken.verify(token, this.serverSecret);
    }

    generate(payload, tokenLifeTime) {
        return jsonwebtoken.sign(payload, this.serverSecret, {
            expiresIn: tokenLifeTime,
        });
    }
}

module.exports = JsonWebToken;
```